Sirena
Um Simulador de Redes Neurais Artificiais

Heitor Barbieri

Sirena
Um Simulador de Redes Neurais Artificiais

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Heitor Barbieri e aprovada pela Comissão Julgadora.

Campinas, 20 de agosto de 1994.

Profa. Heloisa Vieira da Rocha
Orientadora

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

UNICAMP
BIBLIOTECA CENTRAL

Sirena
Um Simulador de Redes Neurais Artificiais[1]

Heitor Barbieri[2]

Departamento de Ciência da Computação
IMECC - UNICAMP

Banca Examinadora:

- Heloisa Vieira da Rocha (Orientadora)[3]
- Márcio Luiz de Andrade Netto[4]
- Ariadne Maria Brito Rizzone Carvalho[3]

---

[1] Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.
[2] O autor é Bacharel em Matemática Aplicada e Computacional - IMECC - UNICAMP.
[3] Professora do Departamento de Ciência da Computação - IMECC - UNICAMP.
[4] Professor do Departamento de Computação e Automação Industrial - FEE - UNICAMP.

Valeu a pena? Tudo vale a pena
se a alma não é pequena.
Quem quere passar além do Bojador
tem que passar além da dor.
Deus, ao mar o perigo e o abysmo deu
mas nelle é que espelhou o céu.

FERNANDO PESSOA

Algumas palavrinhas ...

O mestrado é mais que a especialização em uma área do conhecimento sintetizada em um trabalho escrito, é também um processo de aprimoramento interno.
Partindo-se de um sonho inicial, um objetivo, para que possamos transformá-lo em realidade várias batalhas pessoais têm de ser vencidas.
Quando a poucos passos da reta final chegamos e olhamos para trás, nos apercebemos de que não caminhamos sozinhos.
Nessa jornada muitos foram os que duvidaram e criticaram, e a esses somos gratos pois nos ensinaram que tínhamos que acreditar em nós mesmos, o nosso maior trunfo para vencermos.
Muitos foram os que caminharam ao nosso lado, e a esses somos mais gratos ainda pois a troca de experiências e dificuldades do dia-a-dia foi-nos muito gratificante, fazendo com que crescêssemos ainda mais.
Àqueles porém, que incentivaram, ajudaram e sobretudo acreditaram em nós apenas pela satisfação de nos querer bem, nenhuma palavra será adequada para expressar a nossa alegria.
Não citaremos um nome sequer, pois a ausência de qualquer nome que devesse ser lembrado seria um erro irreparável, portanto, a todos aqueles que de uma forma ou de outra se enquadram em alguma das categorias mencionadas, do fundo do coração ...

O nosso muito obrigado.

HB.
01 de junho de 1994.

Resumo

Rede Neural Artificial (RNA) é um *modelo* que tenta emular uma Rede Neural Biológica.

A área de RNA tem se mostrado bastante promissora, o que pode ser comprovado pela quantidade de trabalhos publicados e de eventos científicos. Mas para que as RNAs atinjam o escopo de aplicações desejado, muitas de suas limitações atuais terão que ser superadas. Ainda não é claro e bem estabelecido o funcionamento das RNAs, não existem metodologias boas e completas para a utilização das mesmas em aplicações, isto é, metodologias que diante de um problema específico a ser resolvido, indiquem qual a topologia de rede, o algoritmo de aprendizagem e a amostragem de informações adequadas ao funcionamento desejado.

Em não se tendo uma metodologia que indique a combinação ótima dos elementos de uma RNA para uma determinada aplicação, resta aos usuários a opção de partir de uma base teórica e, utilizando-se de métodos empíricos, ir formando regras individuais de como conseguir as melhores combinações dos elementos formadores da rede. Esta técnica, porém, apresenta muitas dificuldades em sua realização devido à grande quantidade de variáveis que precisam ser avaliadas durante todo o processo de desenvolvimento da rede.

O presente trabalho busca facilitar o entendimento do funcionamento das RNAs através da familiarização do usuário com os seus elementos formadores.

Foi desenvolvido um simulador de RNAs, denominado Sirena, que através de sua interface gráfica procura minimizar a dificuldade de entendimento dos processos de baixo nível realizados pelas RNAs. Durante o processo de simulação pode-se ter acesso a diversas representações, tanto qualitativas quanto quantitativas, que visam refletir as alterações que ocorrem na rede nas fases de aprendizagem e inferência.

Abstract

Artificial Neural Net (ANN) is a *model* that emulates a Biological Neural Net.

The ANN field has showed very promising which can be verified by the number of published papers and scientific events. In spite, to reach the desired ANN applications scope, many of ANN current limitations have to be overcome since it is not yet and well established the ANN functioning . There is no good and complete methodologies for construct ANN applications, i.e., for a specific problem to be solved, no methodology indicates what the net topology is, the learning algorithm and the sample of information suitable to the desired performance.

If there is no methodology that indicates the better combination of the ANN elements to a specific application, the users have the option to start from a theoretical base and, by using empirical methods, begin constructing personal rules that indicates the better combination of neural elements. The execution of this technique is difficulty because the number of variables that need to be evaluated during the net development process.

The focus of this work is facilitate the understanding of the ANN functioning through the user familiarization with its elements.

A ANN simulator named Sirena was developed and its graphical interface aim to minimize the understanding difficulties of the low level processes executed by ANNs.

During the simulation process one can access to several qualitative and quantitative representations that reflect the net alterations in the learning and inference phases.

Sumário

Lista de Figuras

Capítulo I

Introdução

I.1 - Considerações Iniciais.

O ser humano em toda a sua história tem conseguido solucionar muitos de seus problemas com a construção de ferramentas para auxiliá-lo nas mais diversas tarefas. O que antes eram simples instrumentos que o ajudavam em sua sobrevivência e bem estar, foram crescendo em sofisticação a ponto de se tornarem máquinas que por sua eficiência muitas vezes até o substituem na execução de várias tarefas.

Com a evolução da tecnologia o homem constrói sua máquina mais sofisticada, o computador, e com ela surge a real possibilidade de simular uma característica até então exclusivamente humana: a inteligência.

A construção de uma máquina que realmente possa ser considerada como inteligente ainda pertence ao campo da ficção, embora a sua busca em muito tem contribuído para o avanço da ciência, seja através de discussões filosóficas, seja pelo entendimento de processos psíquicos e biológicos humanos, ou ainda pelo próprio desenvolvimento das ciências exatas.

Neste campo controvertido onde nem mesmo o termo "inteligência" tem um consenso sobre sua definição, parece existir uma concordância de que qualquer sistema inteligente, seja ele humano ou não, tem que ter a habilidade de aprender, pois um sistema que repetidamente comete os mesmos erros dificilmente poderá ser classificado como inteligente.

O estudo da aprendizagem que pode ser definida como a habilidade de se obter entendimentos mais profundos de um tipo de situação que se repete, passou então a ter uma importância muito grande para a construção de um sistema inteligente.

São várias as áreas da ciência preocupadas como a aprendizagem tais como a Filosofia, a Educação e a Psicologia, destacamos, porém, a Inteligência Artificial (IA) que se diferencia das outras por estar fortemente vinculada ao uso do computador para a experimentação de suas teorias.

No que se refere à aprendizagem, a Inteligência Artificial possui dois enfoques para lidar com o assunto, sendo conhecida por IA simbólica ou IA conexionista conforme o enfoque aplicado.

O primeiro enfoque, chamado de IA simbólica, é o mais antigo e busca a aprendizagem em processos de "alto nível", ou seja a nível de conceitos, utilizando-se para isto de mecanismos formais de representação do conhecimento como a lógica formal e também mecanismos provenientes da Psicologia como por exemplo *frames* e redes semânticas.

Esses e outros mecanismos que podem ser encontrados em [Win92] apresentam muitas qualidades mas não são completos o suficiente para promoverem um aprendizado que pode ser considerado geral, não sendo, portanto, considerados soluções definitivas.

Diante da incapacidade destes e outros mecanismos em realizar uma aprendizagem eficiente e principalmente geral, buscou-se em outro extremo a solução para o problema, o enfoque conexionista que busca nos processos biológicos de "baixo nível", isto é, a nível de células nervosas e suas interconexões, a solução para a aprendizagem.

Essa segunda vertente de pensamento, a chamada IA conexionista, embora não tenha a mesma idade da primeira, não é de todo nova já que tem seus primeiros trabalhos publicados na década de quarenta.

Essa área surgiu como a tentativa de modelamento das Redes Neurais Biológicas (RNBs) presentes no cérebro humano, o modelo de máquina inteligente mais desenvolvida que se tem conhecimento.

Dessa tentativa surgiram as Redes Neurais Artificiais (RNAs) que de modelos simples foram se tornando mais poderosas e complexas acompanhando o desenvolvimento das biociências e também dos computadores.

Embora existam pessoas das duas vertentes que defendam que seu lado será capaz, sem a ajuda do outro, de solucionar os problemas da aprendizagem, existem também aquelas que defendem que a solução está a meio caminho dos dois enfoques e que, portanto, será atingida quando ambas atingirem o máximo de seu desenvolvimento.

O caminho seguido pelas RNAs é o de se basear nas RNBs, modelando os neurônios como elementos de processamento (EPs) e suas sinapses por ligações ponderadas indicando a facilidade de transmissão do impulso nervoso de um EP a outro.

São duas as grandes áreas de pesquisa envolvendo as RNAs: uma se preocupa com a definição da topologia da rede, isto é, com a definição dos EPs e de suas interconexões; a outra, com o estudo dos algoritmos de aprendizagem, que são os responsáveis pela aquisição de conhecimento por parte da rede, isto é, eles a "treinam" ou "ensinam" sobre algum domínio de conhecimento.

Os modelos de RNAs por apresentarem qualidades tais como o processamento paralelo e distribuído, memória associativa, tolerância a falhas, degradação graciosa e bom escalonamento de arquitetura, despertam em muitas pessoas o interesse sobre sua definição e seu funcionamento.

Para uma pessoa interessada, porém, leiga no assunto das RNAs, não é intuitivo o entendimento de como os elementos formadores da rede, que isoladamente não desempenham papel importante, conseguem coletivamente realizar o desempenho esperado da rede.

Não são divulgadas, se é que existem, boas analogias para o entendimento de uma RNA como a que existe entre um circuito elétrico e um circuito hidráulico. Todas as tentativas conduzem a analogias fracas e pouco gerais.

São então dois os principais caminhos para o entendimento de uma RNA: a leitura de material escrito sobre o assunto e a visualização de uma rede em funcionamento em um simulador.

Aqueles que pretendem um conhecimento profundo sobre a área têm de optar pela primeira alternativa, mas logo perceberão que o caminho será longo e árduo já que uma profunda incursão pelo campo da Matemática será necessária.

Quanto mais adentramos no estudo de suas teorias e fórmulas, e não são poucas, mais os elementos formadores da rede, que antes agiam de forma milagrosa, passam a fazer sentido.

Nos apercebemos, porém, que mesmo tendo domínio dos mecanismos gerais de uma RNA, a prática exige mais que teoria. Colocarmos uma rede em funcionamento para uma aplicação específica que apresente resultados satisfatórios é uma tarefa muito mais complexa do que se pode inicialmente imaginar.

A saída mais evidente é a procura na teoria por uma solução para a aplicação da mesma na prática. Não existe na literatura uma seqüência bem definida de passos a serem seguidos que seguramente nos levem à escolha de um modelo de rede correta, isto é, um número ideal de elementos de processamento ligados de uma forma ótima, nem a escolha perfeita de um algoritmo de aprendizagem que nos garanta a melhor eficiência da rede para uma determinada aplicação.

Se o estudioso tem pretensões outras além da pesquisa de modelos de redes e de algoritmos de aprendizado novos e mais eficientes, isto é, se pretende ser também um usuário das RNAs, então o ferramental matemático não é suficiente.

O outro extremo, é seguido por aquele que opta pela simulação computacional de uma RNA sem passar pelo embasamento teórico matemático. Este tem tantas chances de obter sucesso quando um músico que toca "de ouvido", e são muitos os que conseguem, mas será sempre uma pessoa cujos procedimentos serão intuitivos, não podendo, portanto, descrevê-los e nem transmiti-los de maneira clara e organizada para as outras pessoas.

A melhor opção que se apresenta no momento é a aquisição de uma pequena, e quanto maior melhor, base teórica, aliada a um bom conhecimento empírico do funcionamento de uma RNA.

Se chegarmos à conclusão que os processos de entendimento e de utilização das RNAs passam por sua simulação computacional, o passo seguinte será a escolha de um simulador adequado.

São vários os existentes, alguns de difícil utilização e outros bastante amigáveis, mas a maioria preocupados com o poder de simulação, isto é, a capacidade de simular redes com um número grande de elementos de processamento, com vários níveis de camadas, várias funções de transferência e de ativação e com a grande velocidade de execução, isto é, quanto maior a velocidade de simulação melhor.

São sem dúvida duas características bastante desejáveis em um simulador, embora ambas pressuponham a existência de um modelo já bem definido por parte do projetista da rede, o que não ocorre na maioria dos casos.

Sem a existência deste modelo a grande velocidade de simulação passa a não ter importância pois ela é realmente relevante somente após a rede estar bem definida. Para possibilitar esta grande velocidade, o simulador pode impor condições ao usuário nem sempre favoráveis ao processo criativo, como por exemplo a imposição da construção da rede em linguagem de programação para a compilação do código antes de sua simulação.

A grande quantidade de recursos oferecidos pelo ambiente de simulação também é outro fator que se não bem gerenciado pode desviar a atenção do usuário do problema inicial, a construção da rede, para a escolha de quais recursos dentre tantos, são realmente necessários em uma primeira etapa e quais poderiam ser deixados para uma etapa posterior.

Tanto o iniciante na área, como aquele possuidor de uma idéia ainda não convertida em uma RNA, precisam, pelo menos em uma primeira etapa, de uma ferramenta voltada para o processo cognitivo da criação de uma RNA e não de uma ferramenta somente voltada para a grande velocidade e poder de simulação.

Nosso trabalho tem como objetivo contribuir neste contexto com a implementação e sugestão de recursos computacionais voltados para facilitar o processo de entendimento e construção de uma RNA.

O sistema Sirena, *Si*mulador de *re*des *n*eurais *a*rtificiais, é uma ferramenta desenvolvida objetivando contribuir com o processo de entendimento e criação de redes neurais.

O termo contribuir é adequado pois muitas questões que surgem neste processo tais como: Quantas unidades a rede deve ter? Como devem ser distribuídas as unidades dentro da rede? Que tipo de informações devem ser oferecidas à rede?, dentre outras, são de difícil solução e cujas respostas definitivas parecem estar longe de serem alcançadas.

A ferramenta é composta de um simulador robusto e de uma interface gráfica que visa facilitar a interação do usuário com o simulador.

Diferentemente de um simulador padrão, o Sirena pretende facilitar ao máximo o seu uso e dar ao usuário a liberdade de realizar grandes modificações em seu modelo de rede neural e de desfazer essas modificações, se assim lhe convier, sem nenhum prejuízo para a simulação.

Dentre as pessoas possivelmente beneficiadas com esse tipo de simulador podemos citar:

- o novato no campo das RNAs que utiliza a ferramenta para a verificação dos aspectos teóricos básicos do assunto;
- o projetista de uma rede neural para aplicações no mundo real que não tendo acesso ao conhecimento matemático envolvido na análise da mesma, utiliza o processo empírico para o desenvolvimento de suas aplicações.
- o estudioso que quer aprofundar seus conhecimentos sobre o funcionamento das RNAs.
- e finalmente o pesquisador teórico que utiliza a ferramenta como um passo intermediário antes da análise quantitativa de um modelo de rede neural.

I.2 - Organização da Tese.

A presente dissertação é composta de sete capítulos organizados da seguinte maneira:

Capítulo I - Introdução.

É o presente capítulo e visa situar o nosso trabalho no contexto das RNAs.

Capítulo II - Redes Neurais.

Este capítulo faz uma introdução ao campo de estudo de Redes Neurais Artificiais, fazendo um paralelo com as Redes Neurais Biológicas (RNBs).

Iniciou-se com uma introdução às Redes Neurais Biológicas, dando desta forma um embasamento mínimo necessário para permitir a analogia entre os dois tipos de redes.

Após esta introdução colocamos o leitor em contato com as RNAs através de sua definição atual juntamente com a descrição de seus elementos formadores.

Uma pequena descrição histórica do assunto dividindo o mesmo em três fases principais, o conexionismo, a fase de desilusão e o neoconexionismo, é feita visando mostrar as dificuldades encontradas em sua evolução.

Terminamos o capítulo descrevendo algumas aplicações já implementadas assim como algumas de suas dificuldades encontradas.

Capítulo III - Aprendizagem.

Neste capítulo salientamos a importância do tópico de aprendizagem nas RNAs.

Mostramos uma das possíveis classificações dos vários algoritmos de aprendizagem existentes e a importância do aprendizado hebiano.

Para que o leitor tenha uma nítida visão de como se processa a aprendizagem em uma RNA, descrevemos detalhadamente o algoritmo de "Backpropagation" utilizado neste trabalho por ser amplamente documentado, discutido e por servir de base para vários outros algoritmos.

Terminamos o capítulo encerrando a parte teórica das RNAs deste trabalho, abordando os problemas envolvendo as RNAs no que se refere à aprendizagem .

Capítulo IV - Objetivo do Trabalho.

Introduzimos neste capítulo a problemática do entendimento e a conseqüente dificuldade em desenvolver e aprimorar modelos de RNAs e seus respectivos algoritmos de aprendizagem.

Diante deste contexto localizamos o nosso trabalho e os seus objetivos.

Capítulo V - Descrição dos Recursos do Sirena e sua Utilização.

Este capítulo tem como objetivo apresentar a interface gráfica do Sirena onde a partir de um exemplo de uma RNA, mostramos cada uma das janelas da ferramenta juntamente com suas respectivas informações.

Além da apresentação das janelas, apresentamos também o funcionamento das caixas de diálogo responsáveis pela maior interação do usuário com a ferramenta.

Encerramos o capítulo resumindo as principais características da interface gráfica assim como sugestões para o seu aprimoramento.

Capítulo VI - Descrição da Implementação da Ferramenta.

Apresenta os aspectos técnicos da implementação do Sirena.

Capítulo VII - Conclusões.

Tecemos neste capítulo alguns comentários sobre as contribuições e limitações da ferramenta, apresentando também algumas propostas para futuras extensões.

Capítulo II

Redes Neurais

Neste capítulo apresentaremos uma introdução às Redes Neurais Biológicas (RNBs) que servirá de substrato à descrição das Redes Neurais Artificiais (RNAs).

Essa descrição será composta pela definição das RNAs e pelo histórico das mesmas desde sua criação até os dias de hoje. Seguirão a essa descrição, algumas áreas de aplicação das RNAs assim como algumas de suas limitações.

II.1 - Redes Neurais Biológicas.

O desenvolvimento da área de **Redes Neurais Artificiais (RNAs)** está todo baseado no conhecimento atual das **Redes Neurais Biológicas (RNBs)**, isto é, redes neurais presentes no homem e em vários animais.

Portanto, para que entendamos as RNAs faz-se necessário um conhecimento mínimo da estrutura e funcionamento das redes biológicas pois as RNAs nada mais são que uma tentativa de modelamento destas redes.

Uma RNB é uma coleção de células especializadas, os neurônios, que estão intensamente interconectados.

No homem, os neurônios estão organizados em diferentes redes neurais, também chamadas de redes nervosas, e que constituem o sistema nervoso. Esse sistema controla desde funções involuntárias do corpo como o equilíbrio e o batimento cardíaco até os mais altos processos de pensamento.

A principal unidade funcional do sistema nervoso é o neurônio, também chamado simplesmente de célula nervosa e sua função é a de receber, transportar e transmitir a mensagem ou **impulso nervoso** a uma grande distância e com acentuada rapidez [Guy76].

Um típico neurônio biológico é constituído basicamente de três partes (veja figura II.1):

- *Corpo celular* - também chamado de *soma*, é nele que se encontra o núcleo da célula responsável pela manutenção da vida da mesma.

- *Dendritos* - são projeções que partem do corpo celular, estendem-se por poucos milímetros e são em número que varia de um a várias centenas. Seu papel é o de transmitir o estímulo nervoso ao corpo celular do neurônio.
- *Axônio* - ou fibra nervosa, é uma projeção do corpo celular (somente um por neurônio) geralmente longa e de diâmetro constante. Ramifica-se intensamente em seu extremo terminal, sendo que as ramificações terminam em pequenas dilatações chamadas **terminais pré-sinápticos** ou **botões sinápticos**.

A função do axônio é de transmitir o impulso nervoso para o exterior do corpo celular do neurônio.

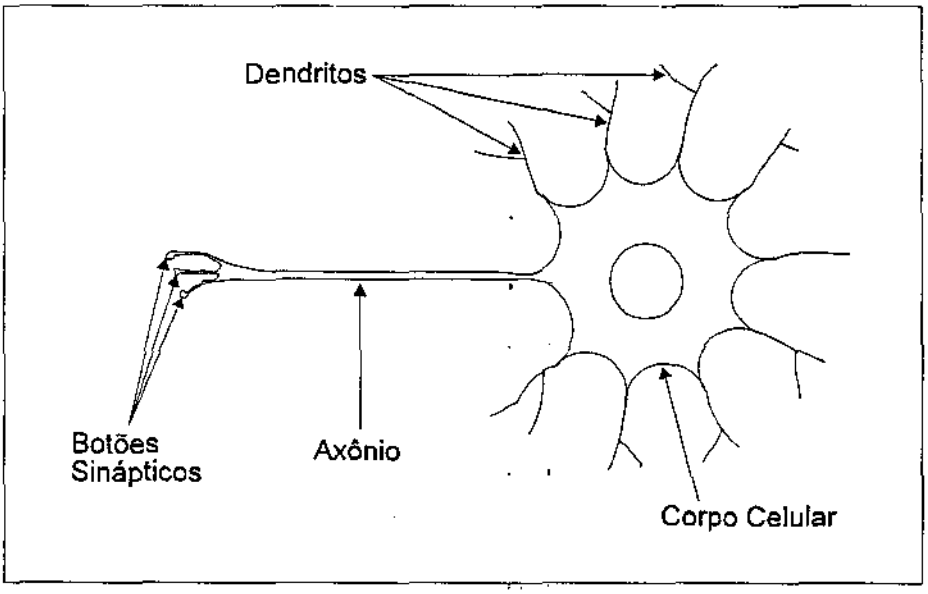


Figura II.1 - Típico neurônio biológico.

A porção do neurônio responsável pela recepção do impulso nervoso é constituída pelos **dendritos** e pelo **corpo celular**, enquanto que a responsável pela transmissão do impulso é o **axônio.**

Para conhecermos o funcionamento de um neurônio é necessário que entendamos os mecanismos de recepção, transporte e transmissão do impulso nervoso. Descreveremos a seguir algumas características desses mecanismos.

II.1.1 - Potencial de Membrana.

Quem desempenha um papel importante nesses mecanismos é a membrana celular do neurônio que tem as mesmas funções que qualquer outra membrana celular exceto por ser especificamente adaptada para a transmissão de estímulos eletroquímicos [Guy76].

Essa transmissão pode ser explicada pela diferença de potencial elétrico entre o interior da membrana e o meio externo do neurônio causada pela passagem de íons de sódio e potássio pela membrana celular do mesmo.

Essa membrana, que é sempre permeável aos íons de potássio, em seu estado normal só permite a passagem de íons de sódio através dela por meio de transporte ativo desses íons, isto é, a célula gasta energia para a passagem desses íons pela membrana.

Esse processo que ocorre no neurônio onde a célula transporta ativamente os íons de sódio do seu interior para o meio exterior (líquido intersticial) é chamado de **bomba de sódio**.

Os íons de sódio por terem carga positiva e por estarem, devido à bomba de sódio, em maior concentração no exterior da membrana, causam uma diferença de potencial na membrana (seu interior fica mais eletronegativo).

Em decorrência desta eletronegatividade, muitos íons de potássio, que também possuem carga positiva, penetram na célula passando pela membrana que lhes é permeável. Como, porém, a saída dos íons de sódio (transporte ativo) é mais rápida que a entrada de íons de potássio (transporte passivo), permanece ainda uma diferença de potencial na membrana.

O potencial eletronegativo criado no interior da célula devido à bomba de sódio é chamado **potencial de membrana**, cuja voltagem real no homem é de cerca de -85mV.

II.1.2 - Onda de Despolarização.

Quando a membrana é excitada com uma intensidade suficiente, seja por meios físicos, químicos ou elétricos, ela se torna no local da excitação altamente permeável.

E como existe o potencial da membrana, os íons de sódio que se encontram em excesso do lado de fora da célula, atravessam a membrana fazendo com que a mesma se torne subitamente positiva no seu interior e negativa no seu exterior. A esse processo chamamos de **despolarização da membrana**.

Ao passarem pela membrana, os íons de sódio acabam por torná-la permeável nas áreas adjacentes às áreas despolarizadas originalmente, permitindo que mais íons de sódio penetrem na membrana.

Esse processo de aumento de permeabilidade da membrana se propaga como uma onda ao longo da membrana sendo, portanto, chamado de **onda de despolarização** ou **impulso nervoso**.

O impulso nervoso então nada mais é do que a propagação da corrente elétrica (movimento das cargas) ao longo da membrana celular do neurônio.

II.1.3 - Repolarização da Membrana.

Imediatamente após o processo de despolarização da membrana, o seu interior fica carregado positivamente o que faz com que a membrana volte a se tornar impermeável à passagem de íons de sódio.

Como o lado exterior da membrana agora está mais eletronegativo, ocorre um fluxo de íons de potássio através da membrana, que lhes é permeável, no sentido do seu interior para o exterior.

Esse processo, chamado de **repolarização da membrana**, normalmente se inicia no mesmo ponto onde se originou a despolarização e se propaga ao longo da membrana de maneira análoga à onda de despolarização.

A repolarização ocorre uns poucos milionésimos de segundo após a despolarização. A membrana não pode transmitir um segundo estímulo nervoso até que a mesma não seja repolarizada.

Depois do processo de repolarização, a bomba de sódio começa a enviar os íons de sódio novamente para fora da célula, o que ocasiona a entrada de íons de potássio para dentro da membrana e a conseqüente restauração da situação inicial.

Deve-se ressaltar, porém, que a bomba de sódio não é necessária para a repolarização da membrana após cada estímulo nervoso (feita pela difusão de potássio para o exterior) devido ao fato de a bomba de sódio trabalhar muito devagar [Guy76].

II.1.4 - Lei do Tudo ou Nada.

Não é qualquer intensidade de estímulo que irá provocar a despolarização da membrana.

Para que a membrana dispare o impulso nervoso, isto é, promova a onda de despolarização, é preciso que o estímulo seja de intensidade forte o suficiente para ultrapassar um patamar mínimo chamado **limiar** do neurônio.

Pelo fato de que estímulos com intensidade abaixo do limiar não produzem impulsos, e que acima desse valor, produzem impulsos que terão sempre a mesma grandeza, independente da intensidade do estímulo, o neurônio então diz-se seguir a **lei do tudo ou nada**.

Devido a esta lei, a informação a respeito da intensidade dos estímulos não fica contida nos impulsos produzidos, mas sim na freqüência de disparo desses impulsos. Estímulos fortes acarretam, portanto, em um aumento na freqüência de disparo dos impulsos nervosos.

II.1.5 - Sinapses.

A união entre os neurônios se faz por uma formação especial chamada **sinapse** (syn= junto e aptein= unir) e é através dela que ocorre a propagação de um impulso nervoso de um neurônio a outro.

Levando-se em conta o sentido da transmissão do impulso nervoso, os constituintes da sinapse são:

- *porção pré-sináptica* - representada pelo axônio do primeiro neurônio.
- *fenda ou fissura sináptica* - espaço de 100 a 500 Å entre os dois neurônios, portanto, um neurônio nunca encosta no outro.

- *porção pós-sináptica* - representada pelos dendritos ou pelo corpo celular do segundo neurônio.

A sinapse, portanto, é a zona de relação entre a porção emissora de um neurônio (axônio) e a porção receptora de outro (corpo celular e dendritos).

Por ter a capacidade de transmitir certos estímulos nervosos e rejeitar outros, a sinapse é um instrumento valioso do sistema nervoso para escolher qual o caminho que o estímulo nervoso deve seguir [Guy76].

A transmissão do estímulo nervoso através da sinapse é feita quimicamente através de substâncias denominadas de substâncias transmissoras.

Todo terminal pré-sináptico (terminação da ramificação do axônio) possui em sua estrutura muitas vesículas que contém tais substâncias que irão determinar a transmissão do impulso nervoso. Conforme a finalidade da substância transmissora ela pode ser classificada em:

- *substância transmissora excitante* - vai causar a excitação da membrana do outro neurônio. Os neurônios que possuem essa substância são chamados **neurônios excitadores**.
- *substância transmissora inibitória* - inibe a excitação da membrana do outro neurônio. Os neurônios com essa substância são chamados **neurônios inibidores**.

O terminal pré-sináptico, que está sempre localizado junto à membrana do corpo celular ou de um dendrito de outro neurônio, quando sofre mudanças momentâneas em sua membrana devido a algum estímulo nervoso, permite que algumas das suas vesículas descarreguem a substância transmissora na fenda sináptica. Esta atua na superfície do outro neurônio de maneira a excitá-lo ou inibi-lo conforme a natureza da substância liberada.

Se a substância transmissora for excitante, ela então provocará uma mudança na permeabilidade da membrana, dando origem, devido à passagem de íons de sódio pela mesma, a um potencial de membrana denominado **potencial excitador pós-sináptico**.

Caso o potencial tenha intensidade suficiente para atingir o valor do limiar do neurônio então causará uma onda de despolarização no axônio, caso contrário nada acontecerá.

Geralmente a estimulação feita por um só terminal pré-sináptico não é suficiente para iniciar um estímulo no axônio. É necessário nesse caso que um grande número de terminais pré-sinápticos atuem ao mesmo tempo para que a quantidade da substância transmissora seja maior, aumentando a permeabilidade da membrana e conseqüentemente o potencial excitador pós-sináptico. Esse processo de soma das atuações dos terminais pré-sináticos é chamado de **somação** [Guy76].

Quando os terminais pré-sinápticos descarregam mas não deflagram uma onda de despolarização, o neurônio torna-se **facilitado**, isto é, mais suscetível a estímulos de outros terminais pré-sinápticos pois as conseqüências químicas dos primeiros terminais ainda podem ser sentidas [Guy76].

Se a substância transmissora liberada por um neurônio for inibidora, seu efeito é o oposto de uma substância transmissora excitadora, pois gera um potencial negativo chamado **potencial pós-sináptico inibidor**.

Enquanto os efeitos dos terminais pré-sinápticos se somam (somação) para que o neurônio atinja seu limiar de disparo, o terminal pré-sináptico inibidor tem o papel de diminuir o potencial da membrana de forma a impedir que o mesmo atinja o valor limiar.

É através desse processo que a sinapse tem a capacidade de impedir que um estímulo nervoso se propague através de um determinado neurônio.

Várias são as características presentes na sinapse, duas, porém, merecem destaque por sua importância nas RNAs.

- *Condução do estímulo em uma única direção* [Guy76]- como somente os terminais pré-sinápticos possuem vesículas com substâncias transmissoras então em uma determinada sinapse a direção do estímulo só pode ser do terminal pré-sináptico para o corpo celular ou para o dendrito de outro neurônio. Essa característica é importante pois permite que os estímulos sejam canalizados na direção desejada.
- *"Memorização da sinapse"* [Guy76] - quando um grande número de estímulos passa através de uma determinada sinapse, ela se torna "permanentemente" facilitada, de maneira que estímulos da mesma origem podem posteriormente atravessá-la com grande facilidade. Acredita-se que seja esse o meio pelo qual ocorre o processo de memorização no sistema nervoso central .

II.2 - Redes Neurais Artificiais.

A seguir definiremos as Redes Neurais Artificiais, descreveremos um pouco de sua evolução e algumas de suas aplicações e limitações.

II.2.1 - Definição.

Kohonen [Koh88] definiu formalmente Redes Neurais Artificiais (RNAs) como redes fortemente paralelas e interconectadas compostas de elementos simples (usualmente adaptativos) e com organizações hierárquicas que se destinam a interagir com os objetos do mundo real da mesma maneira que os sistemas nervosos biológicos.

Uma RNA, portanto, é um *modelo* que tenta emular uma rede neural biológica [Tur92].

Esse modelo, que graficamente é representado por um grafo orientado (veja figura II.2), apresenta os seguintes componentes:

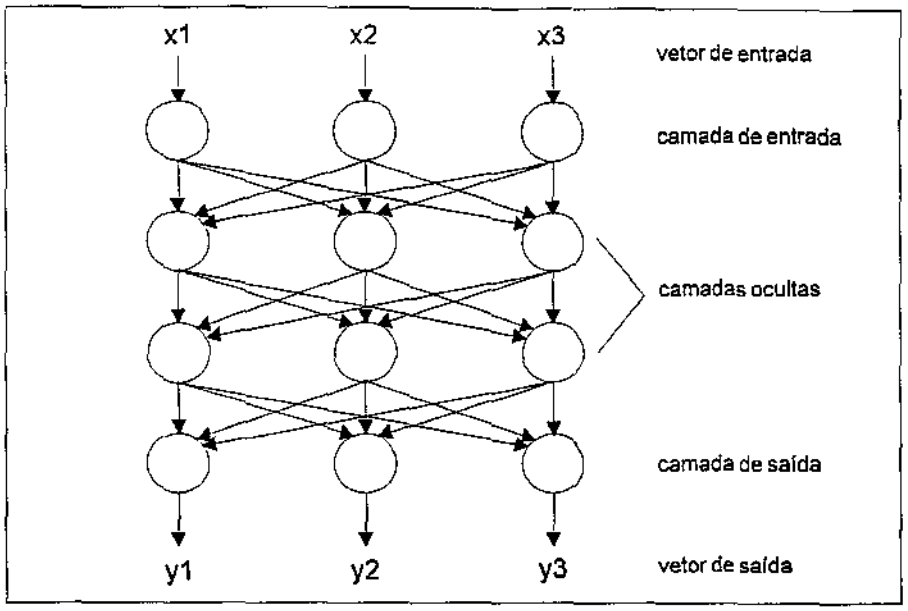


Figura II.2 - Rede Neural Artificial.

- *Elementos de Processamento (EPs)* - são também conhecidos por "neurônios artificiais" pois seu papel é o de modelar o neurônio biológico. São representados no grafo (fig. II.2) através de nós. Cada EP é basicamente constituido de três funções:

  ⇒ *Função de Somação* [Tur92] - como o próprio nome diz, essa função realiza o processo de somação, somando todos os estímulos que chegam ao EP provenientes de outros EPs. Portanto, para um EP_i :

  $$f_{somação\ i} = \Sigma \text{ estímulos que chegam a } i = \Sigma w_{ji} o_j$$

  ⇒ *Função de Ativação* [RHM86a] - esta função determina a grandeza denominada estado de ativação, $a_i(t)$ de um EP_i em um instante *t*, grandeza que representa o estado do EP no instante *t*. A função de ativação, **F**, no caso mais geral depende do estado de ativação anterior e do total de entrada do EP_i , isto é, $a_i(t+1) = F(a_i(t), \Sigma w_{ji} o_j)$, e no caso mais simples é a função identidade onde podemos escrever $a_i(t+1) = \Sigma w_{ji} o_j$. Três funções representativas podem ser vistas na figura II.3.

  ⇒ *Função de Saída* [RHM86a] - é uma função que possuindo o valor de saída da função de ativação, o estado de ativação, calcula a freqüência de disparo deste EP, isto é, $o_i(t) = f_i(a_i(t))$. Embora a função de saída possa ser a função identidade, isto é, $f(x) = x$, ela é na maioria dos casos um tipo de função limiar onde um EP não afeta outro EP a menos que sua ativação exceda a um certo valor.

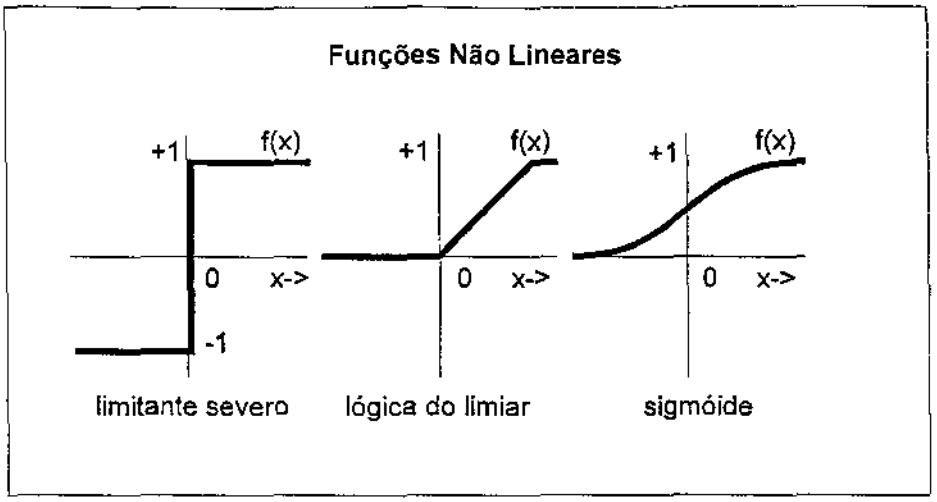


Figura II.3 - Funções de ativação [Lip88].

Os EPs, representados no grafo como nós (fig. II.2), têm então a função de receber e processar a(s) entrada(s) e liberar uma única saída para outros EPs [Tur92].

- *As Ligações* - representam as sinapses em uma RNA pois indicam quais EPs estão se ligando. No grafo uma ligação é representada por um arco orientado indicando qual é o EP "pré-sináptico" e qual é o EP "pós-sináptico".
  Toda ligação em uma RNA tem associada a ela um valor numérico chamado **peso da ligação** que representa a intensidade ou força da ligação (análogo ao processo de facilitação) onde valores positivos indicam intensidades de excitação e valores negativos intensidades de inibição no EP.
  Temos que, se um EP_A está ligado a um EP_B , o estímulo que chega a EP_B é o estímulo enviado por EP_A multiplicado pelo peso da ligação entre eles (P_{AB}) ou

$$\textit{estímulo que chega a B vindo de A} = \textit{saída}_A * P_{AB}$$

- *A Rede* - é composta de uma coleção de EPs, que são agrupadas em camadas, e das ligações conectando esses EPs.
  Existem três tipos de camadas:

  ⇒*A camada de entrada* - é por ela que são fornecidas as informações à rede para que ela as processe.

  ⇒*A camada de saída* - é por ela que a rede devolve as informações processadas, isto é, o valor de saída da rede. É sempre a última camada da rede.

  ⇒*A camada oculta* - pode ser uma ou mais na rede, caracteriza-se por estar situada entre as camadas de entrada e saída razão pela qual também é conhecida como camada intermediária. Seu papel é de auxílio no armazenamento e/ou processamento de informações da rede.

Segundo Hecht-Nielsen [HN88] uma RNA funciona fornecendo a ela um vetor de entrada ou seqüência de números. Cada EP da camada de entrada pega um elemento do vetor, processa esse valor, sempre operando em paralelo com os outros EPs da mesma camada, liberando um único valor de saída (obtido pela função de saída) para os EPs da camada seguinte.

O resultado é um vetor de saída representando algumas características associadas com a entrada. Desde que as entradas e os pesos podem mudar com o tempo, diz-se que a rede se adapta ou "aprende".

As RNAs onde todos os EPs enviam suas saídas para EPs de camadas posteriores e só recebem entradas de EPs de camadas anteriores, isto é, não existem ligações entre EPs de mesma camada e nem ligações direcionadas para EPs de camadas anteriores, são denominadas redes alimentadas para frente ("layered feedforward networks") [RHW86a] .

Denominaremos de *RNA Padrão* toda rede alimentada para frente cujas ligações unam somente EPs de camadas consecutivas, isto é, se uma ligação une um EP da camada i com outro pertencente à camada j então devemos ter a relação $j = i + 1$.

Concluímos a nossa descrição, definindo as duas fases de processamento pelas quais passam os modelos de RNA, a aprendizagem e a inferência, sendo uma durante sua criação e outra durante o seu uso:

- *Aprendizagem* - processo onde de posse das informações de entrada fornecidas à rede, ocorre o ajuste dos valores dos pesos de suas ligações para se ter o desempenho desejado. Nesta fase dizemos que a rede está "aprendendo" ou sendo "treinada".
- *Inferência* - processo onde fornecidas as informações de entrada queremos somente obter as informações de saída. Nesta fase os pesos das ligações são mantidos constantes.

II.2.2 - Histórico.

A história das Redes Neurais Artificiais (RNAs) pode ser dividida em três fases: a primeira delas, chamada conexionismo, iniciou-se na década de 40 e permaneceu até o temporário arrefecimento na área provocado por Minsky e Papert [PM69]; a segunda fase que abrangeu as décadas de 60 e 70 se caracterizou por um descrédito na área devido à verificação de limitações aparentemente intransponíveis das redes. A terceira e última fase é chamada de neoconexionismo, e representa o renascimento da área que começou na década de 80 e prossegue até os dias de hoje [CS88].

II.2.2.1 - Primeira Fase - Origens.

A primeira fase ou origens do conexionismo[1] abrange as décadas de 40 e 50 quando os cientistas estavam motivados pela noção de que os neurônios eram chaves liga-desliga (dispara-não dispara) da mesma forma que os computadores digitais tinham "bits" que estavam ligados (1) ou desligados (0) [Lev89].

A motivação por detrás desse primeiro esforço era a de conciliar as teorias sobre computação e modelagem de circuitos binários com as recentes descobertas em neurobiologia e a fascinação com o comportamento inteligente [Ten90].

Podemos destacar alguns dos acontecimentos e pesquisadores mais importantes na área nessa fase.

Warren S. McCulloch e Walter H. Pitts (1943) -Foram os primeiros teóricos a conceber os fundamentos da computação neural. Com a publicação de um artigo em 1943 [MP43], iniciaram essa área de pesquisa nos anos 40 [Koh88].

Em seu artigo [MP43] McCulloch e Pitts aplicaram a lógica simbólica ao problema de descrever o que redes neurais podem fazer, provando que todos os processos que puderem ser descritos com um número finito de expressões simbólicas (por exemplo aritmética simples, classificação e aplicação recursiva de regras lógicas) poderiam ser incorporadas em redes formadas por neurônios formais, também chamados de unidade MP.

Os neurônios formais são chaves lógicas simples onde cada neurônio só pode computar funções lógicas simples como x *AND* y, x *OR* y, NOT x, x AND NOT y. São, portanto, simplificações extremadas do funcionamento de um neurônio real. As redes formadas por tais neurônios são denominadas redes McCulloch-Pitts.

Apesar dessas simplificações, as redes McCulloch-Pitts, e não só uma unidade MP isolada, são importantes por poderem incorporar quaisquer operações e processos que puderem ser descritos em termos lógicos [CS88].

O neurônio formal (veja figura II.4), em relação a um neurônio biológico apresenta as seguintes semelhanças:

- *sinapses* - aqui representadas por ligações (arcos ligando dois nós que representam as unidades) que possuem pesos associados a elas indicando numericamente o quanto as sinapses são excitatórias ou inibitórias.
- *freqüência de disparo* - nas unidades MP, representadas aqui por círculos ou triângulos, a freqüência de disparo ou é zero (não houve disparo) ou é um (máxima freqüência de disparo).
- *valor limiar* - representado pelo valor obtido da multiplicação da freqüência, que é sempre um, da unidade *t* ("threshold") pelo valor do peso da ligação correspondente.
- *somação* - a unidade MP soma os valores de saída das outras unidades (freqüência * peso da ligação). Se o valor obtido (total de entrada) adicionado ao valor do limiar for

[1] Muitos autores utilizam o termo conexionismo como sinônimo de RNAs não importando qual a fase histórica.

zero ou mais, significa que o total de entrada (potencial excitador) atingiu o limiar e portanto, a freqüência de saída da unidade é um. Se o valor não atingiu o limiar (soma negativa), a freqüência de saída da unidade é zero, isto é, ela não disparou.

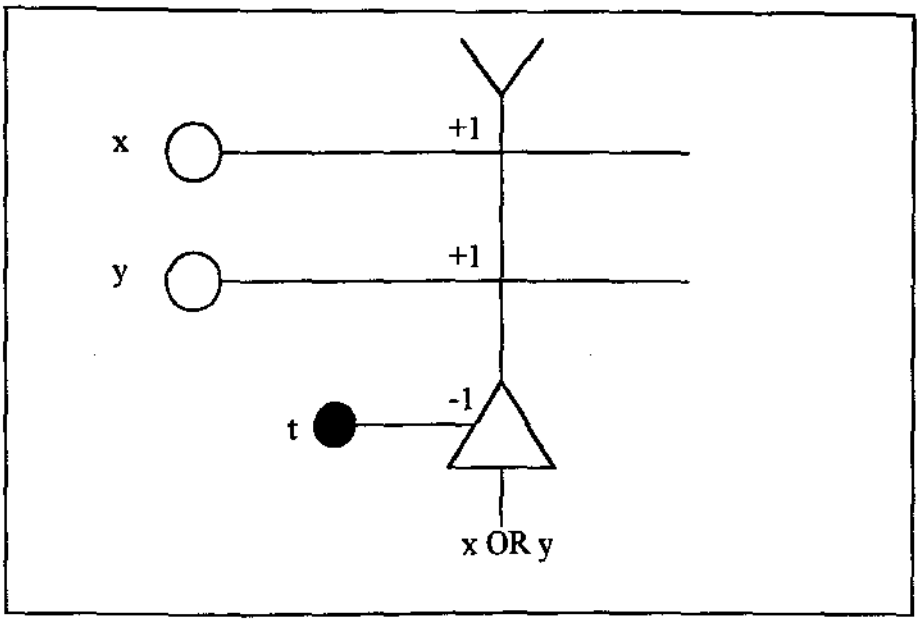


Figura II.4 - Neurônio MP.

Donald O. Hebb (1949) - O principal problema após McCulloch e Pitts era entender como uma rede neural poderia aprender, isto é, armazenar o conhecimento necessário para se obter o desempenho desejado da rede.

Acreditava-se na época que eram necessárias algumas mudanças físicas na rede para que esta suportasse o aprendizado, embora não estivessem claras quais deveriam ser estas mudanças.

Hebb em seu trabalho "Organization of Behavior" [Heb49] propôs que uma mudança razoável e biologicamente plausível poderia fortalecer as conexões entre os elementos de uma rede somente quando as unidades pré e pós sinápticas estivessem ativas simultaneamente [RM86].

Em outras palavras quando uma unidade A e uma unidade B estão simultaneamente excitadas, cresce a força da conexão entre elas. Podemos dizer que o ajuste do peso de uma ligação é proporcional ao produto dos valores de saída dos neurônios ligados:

$$\boxed{\Delta\omega_{i,j} = Ky_i y_j}$$

onde w_{ij} é o valor do peso da ligação dos neurônios i e j com valores de saída y_i e y_j respectivamente.

Nessa formulação se o produto é positivo a mudança faz a conexão mais excitatória e se o produto é negativo, a mudança faz a conexão mais inibitória [RHM86].

Um dos grandes méritos das idéias de Hebb, apesar da ausência de uma evidência definitiva para suportá-las, foi o de ter instigado na época muitas investigações sobre o aprendizado em RNAs [CS88].

II.2.2.2 - Segunda Fase - Desilusão.

Essa fase abrange as décadas de 60 e 70 quando a pesquisa de IA moveu-se em direção aos programas de processamento simbólico para a realização e explicação de tarefas cognitivas simples, com pouca referência a como os animais vivos realizam tais tarefas [Lev89].

Foi uma época de desilusão no campo das RNAs pois tornou-se evidente que as limitações de uma única camada e separabilidade linear nas redes impediriam que os algoritmos de aprendizagem fossem capazes de solucionar mesmo problemas simples [Ten90].

Seymour A. Papert e Marvin L. Minsky (1969) - em seu livro "Perceptrons" [PM69], provaram que perceptrons de uma única camada só resolveriam problemas cuja solução fosse linearmente separável, isto é, a região de solução do problema pudesse ser separada (limitada) por um hiperplano (veja figura II.6).

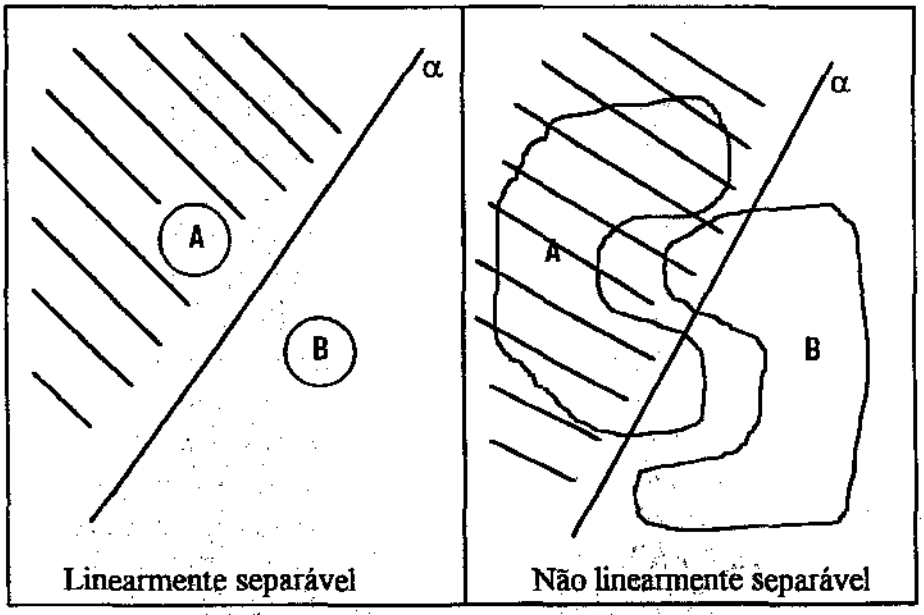


Figura II.6 - Região de solução A sendo delineada por um hiperplano.

Com isso eles provaram que a mais simples arquitetura de rede MP que implementa a função x *OR ELSE* y [2], precisa de duas unidades MP ocasionando uma situação complicada, pois a função *x OR ELSE y* é computacionalmente universal no sentido de Turing, isto é, todas as outras funções podem ser expressas como combinações de *NOT*(x *OR ELSE* y) [CS88].

Para solucionar esse problema seria necessário um perceptron com mais de uma camada de unidades MP, isto é, a saída de uma unidade MP se ligaria à entrada de outra unidade MP e não mais a uma unidade "motora". Só que perceptrons multicamadas apresentaram o *problema de atribuição de crédito* [RC90], ou seja, o problema de como treinar os pesos de suas ligações .

[2] x OR ELSE y é equivalente a (x AND NOT y) OR (y AND NOT x).

A suposta ausência de solução para este problema foi o maior argumento usado por Minsky e Papert para desacreditar os sistemas de aprendizagem e conseqüentemente toda a área das RNAs [RC90].

II.2.2.3 - Terceira fase - Neoconexionismo.

Esta fase que se iniciou na década de 80 e continua até os dias de hoje é caracterizada pelo restabelecimento da pesquisa em redes neurais como uma atividade promissora.

Iniciou-se lidando com questões importantes tais como o estudo comparativo entre o comportamento da rede e o comportamento energético de outros sistemas dinâmicos e o projeto de redes de aprendizagem não lineares com um número fixo de multicamadas arbitrariamente conectadas[Ten90].

Dentre o grande número de acontecimentos e pesquisadores do neoconexionismo destacaremos dois:

John J. Hopfield (1984) - demonstrou uma analogia formal entre uma rede com conexões simétricas composta de elementos semelhantes a neurônios e uma nova classe de materiais magnéticos chamados "spin glasses". À esta rede denominou-se rede de Hopfield (veja figura II.7).

Hopfield utilizando uma analogia entre as propriedades computacionais de coleções de neurônios simples a as propriedades termodinâmicas de coleções de partículas mostrou que da mesma maneira que as leis da termodinâmica são insensíveis aos detalhes das forças entre partículas, as habilidades computacionais das redes neurais são insensíveis à mudança de detalhes no seu modelo [Fir88].

Observou que o comportamento da rede é governado por uma função de energia global e que a modificação dos valores dos pesos das ligações poderia diminuir a energia do sistema levando a configurações estáveis que poderiam ser usadas para armazenar informações [CS88].

Outras contribuições de Hopfield foram [Fir88]:

- Um estudo sistemático de memórias endereçáveis por conteúdo usando uma rede de neurônios com propriedades simples.
- A proposição de um circuito elétrico linear (circuito de Hopfield) para a solução de equações diferenciais de neurodinâmica clássica. Tal circuito forneceu o protótipo para a implementação subseqüente de redes neurais em circuitos VLSI.
- A solução de dois problemas não biológicos de complexidade combinatória através do uso de RNA: o conversor analógico/digital e o problema do caixeiro viajante.

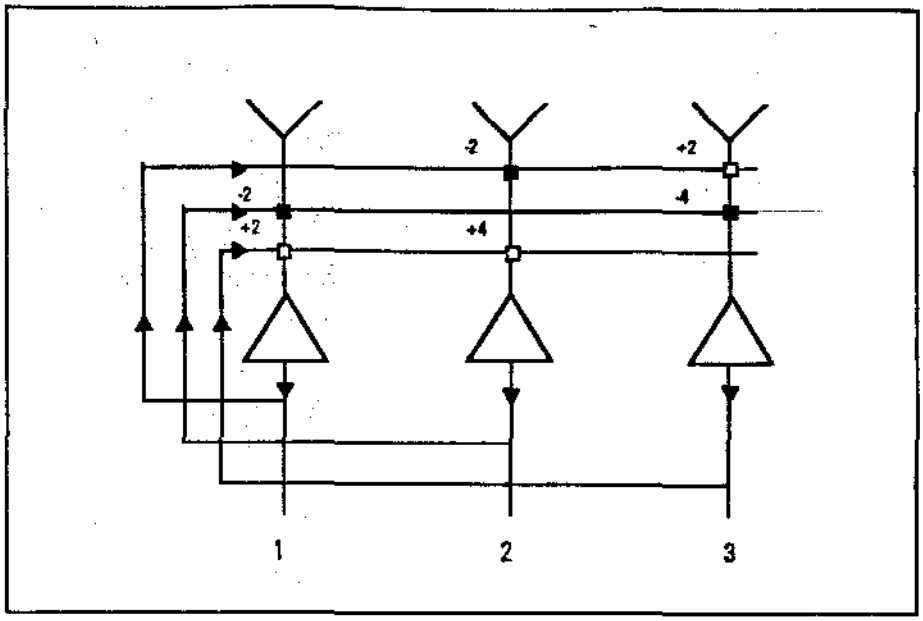


Figura II.7 - Rede de Hopfield.

Possivelmente a maior colaboração de Hopfield foi o auxílio no restabelecimento da credibilidade das RNAs, mostrando que as mesmas poderiam ser utilizadas para problemas reais com considerável complexidade e não somente para problemas artificiais ("toy problems").

Hopfield, porém, por não utilizar os perceptrons acabou por não solucionar a questão da aprendizagem dos mesmos, justamente a causa do arrefecimento na área das RNAs. A solução de tal problema foi conseguida pelo outro destaque no neoconexionismo, o grupo PDP.

McClelland e Rumelhart (1982) - James McClelland da Universidade de Carnigie-Mellon e David Rumelhart da UCSD formaram em 1982 um grupo de estudos multidisciplinar que incluía pessoas especializadas nas áreas de Física, Matemática, Neurociência, Biologia Molecular, Ciência da Computação e Psicologia objetivando explorar as implicações dos modelos de redes neurais e descrever suas conclusões em um livro [RM86].

O grupo se denominou *Processamento Paralelo e Distribuído* (PDP), nome que enfatizava a natureza paralela do processamento, o uso de representações distribuídas, de controle distribuído e também de sistemas de processamento geral [RM86].

Em 1986 foram publicadas as conclusões do grupo em um livro composto de dois volumes, "Parallel Distributed Processing" [RM86][RM86a], que se tornou um dos livros de consulta mais importantes para os estudiosos da área.

Foi do grupo PDP que surgiu o algoritmo de "Backprogation"[3] [RHW86] [RHW86a] solucionando o problema da atribuição de créditos para perceptrons multicamadas. Por sua importância histórica e também por ser um dos algoritmos de aprendizagem mais difundidos na

[3] Embora a popularização do algoritmo de "Backpropagation" se deva ao grupo PDP, variantes deste procedimento já haviam sido descobertas independentemente por Werbos [Wer74], Parker [Par85] e Le Cun [Cun85].

área, apresentaremos um descrição mais detalhada do mesmo no Capítulo III que trata especificamente do processo de aprendizagem.

II.2.3 - Aplicações.

Dentre as áreas que têm sido sugeridas e efetivadas como potencial para a aplicação de RNAs, a mais importante é a de *Reconhecimento de Padrões*.

O termo reconhecimento de padrões surgiu no início da década de 60 para descrever tarefas como a detecção de formas simples, como caracteres manuscritos e mapas meteorológicos. O objetivo, porém, sempre foi o de chegar à *percepção artificial*, ou seja, imitar as funções sensoriais dos sistemas biológicos [Koh88].

Nas primeiras tentativas foram usadas redes neurais elementares como Perceptron [Ros58] e Adaline [WH60] mas logo percebeu-se que o desempenho dos sistemas sensoriais biológicos é muito difícil de ser alcançado, haja visto que a maioria desses sistemas não opera automaticamente e sim associado a processos cognitivos globais [Koh88].

A solução encontrada no fim da década de 60 foi a de adotar aplicações cujas soluções "artificiais" fossem mais efetivas que as encontradas por sistemas naturais.

Atualmente as áreas de aplicação mais importantes para o reconhecimento neural de padrões são:

- sensoriamento remoto.
- análise de imagens médicas.
- visão computacional na área industrial.
- dispositivos de entrada para computadores.

Outras tarefas concretas para as quais computadores especiais já foram desenvolvidos são:

- segmentação e classificação de regiões a partir de imagens.
- reconhecimento de caracteres manuscritos e texto.
- reconhecimento de voz.
- processamento e recuperação de imagens com "ruído".

Tomada de Decisão é outra área de utilização das RNAs que abrange aplicações cuja estratégia de solução dos problemas envolvidos é composta por regras que só podem ser descritas de maneira implícita. Um método prático de tais regras serem aprendidas é através da apresentação de exemplos onde os problemas foram solucionados. Como essa é a maneira natural de aprendizagem de uma RNA, muitos modelos de redes têm sido desenvolvidos para a solução dos problemas envolvidos em tais aplicações. Dentre essas aplicações podemos citar [Tur92]:

- *Serviços financeiros* - identificação de padrões em informações do mercado financeiro.

- *Avaliação de pedidos de empréstimo* - julgamento da idoneidade do pedido de empréstimo baseado em padrões de informação de pedidos anteriores.
- *Diagnóstico médico* - treinamento de redes neurais com casos anteriores de pacientes.
- *Seqüenciamento de DNA* - análise de padrões em estruturas de DNA e rápida comparação de padrões em novas seqüências.

Uma outra área de potencial aplicação das RNAs é a de *Aprendizado em Robôs Inteligentes* [Koh88]. Os robôs inteligentes são aqueles dos quais se espera que planejem suas próprias ações, como em tarefas de montagem onde as peças se localizam em locais aleatórios, sendo uma característica vital em tais robôs o seu alto grau de aprendizagem. Como tais robôs estariam sujeitos a situações com alto grau de entropia, o aprendizado necessário dificilmente poderia ser formalizado através da programação lógica ou procedural tornando o uso das RNAs se não necessário, pelo menos muito adequado a este tipo de situação.

II.3 - Considerações Finais.

Muitos foram os avanços conseguidos no campo das RNAs desde as primeiras tentativas de modelamento e simulação dos neurônios biológicos.

A quantidade de trabalhos publicados e de eventos científicos na área indicam que a mesma é promissora e está se desenvolvendo rapidamente, mas para que as RNAs atinjam sua maturidade teórica e ampla difusão tal como a obtida pela computação tradicional, muitas das suas limitações atuais terão de ser superadas.

As dificuldades das RNAs citadas em [Ten90] [Tur92] [Win92], começam pela escolha do tipo de problema apropriado a elas, pois existem problemas, tais como processamento aritmético e de dados, onde os computadores convencionais apresentam um desempenho superior.

Para as RNAs os problemas mais adequados são aqueles de difícil solução para as técnicas convencionais baseadas em regras, ou seja, para aqueles que apresentam um alto grau de entropia ou variabilidade.

Problemas que envolvam o manuseio de aspectos temporais de informações fornecidas devem ser evitados pois os métodos existentes para tal nas RNAs ainda se encontram em estágio de pesquisa.

Aplicações que exijam o caminho seguido para a solução do problema como no caso de um diagnóstico médico também devem ser evitados, pois a rede treinada não deixa de ser uma caixa preta de pesos e conexões revelando pouco sobre a estrutrura do problema, pois ela não dispõe de facilidades de explanação.

No processamento simbólico, paradigma diferente do das RNAs, todo o conhecimento é representado de maneira clara e em local bem definido e diferente do local onde estão armazenados as regras para a manipulação desse conhecimento. Neste paradigma podemos sempre explicar como o sistema chegou a uma determinada conclusão, podemos também verificar o processo de aprendizagem seja pelo aumento da base de conhecimento, seja pelo aumento do número de regras (máquina de inferência) utilizadas para a manipulação da base de conhecimento.

Em um modelo de RNA, ao contrário, todo conhecimento está armazenado nos pesos das ligações que fazem parte do mecanismo de inferência da rede. Portanto, o conhecimento e o mecanismo de manipulação desse conhecimento estão aglutinados, impedindo desta maneira que se veja quais os critérios e quais as informações que foram utilizados na obtenção das informações de saída obtidas pela rede.

Determinado o problema adequado, uma dificuldade freqüentemente encontrada é a da determinação a priori da complexidade do problema e a conseqüente dificuldade do projeto da topologia da rede. Esse problema envolve questões como as formuladas por Winston [Win92]:

Que conceitos intermediários são construídos pelas unidades ocultas? onde por conceito intermediários entende-se características de alto nível obtidas das informações de entrada para a solução do problema proposto.

Como as unidades de saída da rede usam esses conceitos intermediários para alcançarem as respostas finais da rede?

Como usar as entradas da rede para expressar o que se sabe e como usar as saídas da rede para expressar o que se quer saber?

Quantas unidades a rede deve ter e como devem ser distribuídas tais unidades dentro da rede?

Todas as perguntas são de difícil resposta principalmente porque a informação utilizada na solução dos problemas é codificada na rede com valores reais através dos pesos das ligações, as representações dos conceitos são distribuídos dentre as várias unidades de processamento e porque tais unidades tipicamente possuem muitas conexões de entrada, isto é, várias ligações que chegam a cada unidade.

Definida a topologia, a fase seguinte é a fase do treinamento ou aprendizagem da rede acrescentando mais questões às já existentes. Tais questões serão discutidas no Capítulo III destinado ao assunto de aprendizagem em RNAs.

Na fase de inferência, onde a rede está adequada ao problema e os problemas de aprendizagem supostamente já superados, o maior problema são as limitações e custo da tecnologia atual de equipamentos paralelos. Esse problema restringe a maioria das aplicações a simulações computacionais em equipamentos convencionais, diminuindo em muito a eficiência da rede a ponto de tornar muitas aplicações, devido à quantidade de processamento exigido, impraticáveis.

Por não se ter ainda respostas a essas questões e problemas formulados, Winston [Win92] afirma que o desenvolvimento com sucesso da tecnologia das redes neurais exige ainda muito tempo e experiência. Conclui que os especialistas em RNAs são artistas e não meros seguidores de manuais pois não existe uma metodologia definida para o desenvolvimento de uma rede neural para um problema específico, sendo a maioria do conhecimento obtida através de tentativa e erro.

Capítulo III

Aprendizagem

Neste capítulo faremos uma pequena introdução ao assunto de aprendizagem em Redes Neurais Artificiais (RNAs) salientando algumas das suas principais características.

Apresentaremos a regra de aprendizagem utilizada pela maioria dos paradigmas de aprendizagem em RNAs, a Regra de Hebb, e a classificação mais conhecida dos algoritmos de aprendizagem na área (Níveis de Supervisionamento).

Para dar uma idéia melhor do funcionamento de um algoritmo de aprendizagem, mostraremos em detalhe o algoritmo de Retropropagação do Erro ("Backpropagation") escolhido por ser muito difundido entre os algoritmos de aprendizagem supervisionada e por ter dado origem a vários outros algoritmos de aprendizagem.

Terminaremos o capítulo apresentando os problemas mais conhecidos encontrados no processo de aprendizagem e também algumas conclusões em torno do assunto.

III.1 - Introdução.

A história indica que a fascinação humana pelo desenvolvimento de máquinas inteligentes tem sua origem desde a civilização grega antiga onde explanações teóricas sobre o cérebro e processos mentais foram sugeridos por filósofos como Platão (427-347 AC) e Aristóteles (384-322 AC) [Koh88].

Qualquer sistema inteligente, porém, seja ele humano ou não, tem que ter a habilidade de aprender, pois um sistema que repetidamente comete os mesmos erros dificilmente poderá ser classificado como inteligente [Ster90].

Uma definição sobre o ato de aprender é a habilidade de obter entendimentos mais profundos de um tipo de situação que se repete [CM85]. Quanto melhor for o nosso entendimento sobre um determinado assunto, menores serão as chances de cometermos erros em coisas relacionadas a ele.

Uma outra definição interessante sobre o ato de aprender é a de ganhar conhecimento sobre um assunto ou habilidade em uma determinada atividade através da experiência ou por meio de ensino [Dav87].

Estas definições nos revelam algumas características interessantes relacionadas com a aprendizagem:

A vinculação com o conceito de desempenho - freqüentemente espera-se que o processo de aprendizagem, seja este voluntário ou não, leve a um melhor desempenho em uma atividade qualquer (diminuição do número de erros cometidos).

A aquisição de conhecimento - o conhecimento adquirido pode ser resultado somente do processamento das informações já existentes (entendimento mais profundo) ou também da aquisição de novas informações (ganhar conhecimento).

A forma de aquisição de conhecimento - no caso em que novos conhecimentos estão relacionados à aquisição de novas informações, esses conhecimentos podem ser ensinados ou simplesmente adquiridos (experiência).

Estas características presentes no processo de aprendizagem também serão aplicadas à aprendizagem das RNAs (medida do erro obtido, aquisição de novos conhecimentos, aprendizado supervisionado e não supervisionado).

Embora existam várias formas, definições e teorias sobre aprendizagem, estaremos neste capítulo interessados somente na aprendizagem em Redes Neurais Artificiais (RNAs).

Como todo conhecimento nas RNAs é codificado através dos pesos atribuídos às conexões entre os nós (elementos de processamento), a aprendizagem nas mesmas é realizada pela mudança desses pesos [Die90].

Portanto, o objetivo principal da pesquisa em RNAs é descobrir procedimentos eficientes de aprendizagem que permitam a essas redes construírem representações internas, complexas e distribuídas do conhecimento a ser adquirido.

Os procedimentos de aprendizagem precisam ser capazes de modificar os pesos da rede de tal maneira que unidades "ocultas", isto é, unidades que não pertençam nem a camada de entrada nem a de saída, venham a representar importantes características necessárias à tarefa a ser desempenhada pela rede [Hin90].

O processo de aprendizagem geralmente começa com a rede sem nenhum conhecimento armazenado, isto é, os seus pesos possuem valores aleatórios. Com a constante e cíclica apresentação das informações de treinamento, a rede através de um algoritmo de aprendizagem vai modificando seus pesos até o ponto onde não é mais necessária a apresentação de informações, pois de alguma forma o conhecimento representado por estas já está codificado e armazenado na rede.

A partir desse ponto, como não é mais necessária a apresentação de informações de treinamento à rede, diz-se que a fase de aprendizagem terminou. A rede então está pronta para a próxima fase, a fase de inferência, onde ela passa a se comportar de maneira análoga a uma função matemática, fornecendo para cada informação de entrada somente uma informação de saída e que é sempre a mesma para aquela informação de entrada.

Existem tarefas, porém, em que a rede precisa estar aprendendo constantemente, por exemplo uma rede que controla a altitude de um artefato espacial cuja massa decresce à medida que o combustível é consumido [HN88]. Nesses casos podemos pensar que a fase de inferência mescla-se com a fase de aprendizagem, pois as informações de treinamento são as mesmas utilizadas pela aplicação.

A maneira como a rede vai modificando seus pesos durante a fase de aprendizagem depende exclusivamente do algoritmo de aprendizagem por ela utilizado, sendo que a vasta maioria dos

modelos de RNAs utiliza alguma variação da regra de Hebb, analisada na próxima seção, como função de modificação do peso das conexões [Ster90].

A técnica de aprendizagem descrita até agora é caracterizada pela mudança somente de peso, isto é, todo o processo de aprendizagem é feito alterando-se os valores dos pesos mas deixando-se a topologia da rede inalterada.

É possível, porém, e às vezes até necessária, a utilização de técnicas que modifiquem a estrutura da rede, sendo dois os métodos disponíveis [Die90]:

Os procedimentos de aprendizado que fisicamente modificam a topologia da rede - durante o processo de aprendizagem neurônios (EPs) que não existiam inicialmente podem ser criados para o armazenamento de conceitos.
Técnicas de aprendizagem que modificam fisicamente a estrutura da rede são raras pois assume-se que a estrutura completa está disponível desde o nascimento, como no caso de seres vivos e que, portanto, modificações da estrutura são degenerações da mesma.

Os procedimentos ditos de recrutamento - nesses procedimentos a rede é formada por duas classes de unidades: as comprometidas, unidades que já representam alguma informação ou conceito, e as unidades denominadas livres.
A aprendizagem de recrutamento é o fortalecimento de conexões entre um grupo de unidades comprometidas e uma ou mais unidades livres, resultando na transformação de unidades livres em unidades comprometidas à medida em que novas informações são adquiridas pela rede.

De uma maneira geral podemos dizer que são três os fatores que influenciam o aprendizado em uma RNA [Ster90]:

- *O ambiente* - representado na rede pelas informações de entrada fornecidas. Conforme as características dessas informações e a forma com que elas são apresentadas à rede o processo de aprendizagem pode se comportar de maneira diferente.
- *Restrições estruturais da arquitetura do sistema* - a quantidade de EPs, de ligações e a distribuição dos mesmos na rede aliada aos tipos de funções utilizadas pelas EPs.
- *Regras de aprendizagem empregadas* - onde cada regra utiliza uma maneira diferente de aprendizagem.

III.2 - Aprendizado Hebbiano.

Na década de 40 a aprendizagem possuía grande importância pois estava unicamente associada às propriedades dos cérebros animais, sendo, portanto, de difícil entendimento como qualquer coisa que se assemelhasse a uma rede neural artificial pudesse também aprender [RZ86].

Acreditava-se na época que alguma mudança física deveria ocorrer em uma rede neural para suportar a aprendizagem, mas ainda não estava claro como tal mudança deveria ocorrer.

Em 1949 Donald Hebb [Heb49] publicou "The Organization of Behavior" no qual introduziu várias hipóteses sobre os substratos neurais da aprendizagem e da memória incluindo a regra de aprendizagem que levaria o seu nome. [SS90]

A regra de Hebb diz que "quando um axônio da célula A está próximo o suficiente para excitar a célula B e repetidamente ou persistentemente toma parte no seu disparo, algum processo de desenvolvimento ou mudança metabólica ocorre em uma ou mais células tal que a eficiência de A, como uma das células que disparam B, é incrementada".

Isto implica que um grupo de neurônios fracamente conectados, se ativados sincronamente, tende a se organizar em formações mais fortemente conectadas. [Vem88].

A hipótese de Hebb era baseada em evidências do condicionamento clássico da psicologia sem qualquer evidência fisiológica, muito embora exemplos de processos neurais consistentes com sua hipótese foram descobertos depois de 1949, primeiro em invertebrados e depois em vertebrados [Lev89].

A essência das idéias de Hebb persiste até hoje em vários paradigmas de aprendizagem onde os detalhes das regras para a mudança dos pesos pode variar mas a idéia essencial de que a força das conexões entre as unidades deve mudar em resposta a alguma função da atividade correlacionada das unidades conectadas ainda domina os modelos de aprendizagem [RZ86].

Pode-se formular de modo geral a *Regra de Hebb* através das seguintes equações:

Seja um neurônio A com freqüência de disparo $o_A(t)$ ligado a outro neurônio B com freqüência de disparo $o_B(t)$ através de uma ligação com peso $\omega_{A,B}(t)$.

Em modelos lineares a freqüência de disparo de B devido a A é dada pelo produto de $o_A(t)$ por $\omega_{A,B}(t)$ enquanto que em outros modelos a relação entre as entradas e saídas pode não ser linear [SS90].

Hebb afirma que a alteração do valor do peso é dependente tanto da atividade de A quanto da atividade de B.

Uma maneira bastante estendida de se expressar a regra de Hebb pode ser vista através da seguinte expressão [RHM86a]:

$$\Delta\omega_{A,B} = g(a_B(t), t_B(t)).h(o_A(t), \omega_{A,B}) \qquad (1)$$

Onde:

g e h são duas funções genéricas,
$t_B(t)$ é uma informação de ensino que se fornece ao neurônio B,
$a_B(t)$ é o estado de ativação do neurônio B em um instante t e
$o_B(t)$ é o valor de saída do neurônio B, tal que $o_B(t) = f(a_B(t))$

Nas versões mais simples do aprendizado hebbiano, não existe informação de ensino e as funções g e h são proporcionais a seus primeiros argumentos, isto é,

$$\begin{aligned} g(t) &= K_1 a_B(t) \quad \text{e} \\ h(t) &= K_2 o_A(t) \end{aligned} \tag{2}$$

resultando em

$$\Delta\omega_{A,B} = \varepsilon\, a_A(t).o_B(t) \tag{3}$$

que é a maneira mais conhecida e direta da regra de Hebb, onde ε é uma constante numérica de proporcionalidade, geralmente de pequeno valor e que representa a taxa de aprendizagem.

A regra denominada de Regra Delta e que é uma generalização da regra de aprendizagem para um perceptron pode ser obtida de (1) fazendo-se:

$$\begin{aligned} g\big(a_B(t), t_B(t)\big) &= t_B(t) - a_B(t) \quad \text{e} \\ h\big(o_A(t), w_{A,B}(t)\big) &= o_A(t) \end{aligned} \tag{4}$$

resultando na seguinte forma:

$$\Delta\omega_{A,B} = \varepsilon\big(t_B(t) - a_B(t)\big)\, o_A(t) \tag{5}$$

III.3 - Classificação dos Algoritmos de Aprendizagem.

Os algoritmos de aprendizagem das RNAs podem ser classificados de várias maneiras conforme características da rede (funcionais/topológicas) e da forma de aprendizagem a ser ressaltada.

Mostraremos a seguir a classificação destes algoritmos freqüentemente considerada como a mais importante [Die90].

Essa classificação, também conhecida como Interação com o Ambiente ou também como Níveis de Supervisionamento, salienta a maneira como uma RNA é treinada na fase de aprendizagem, sendo duas as possíveis formas: o aprendizado supervisionado e o aprendizado não supervisionado.

III.3.1 - Aprendizado supervisionado.

Nessa forma de aprendizado, um padrão de entrada, representado no vetor de entrada da rede, é apresentado juntamente com um padrão de saída desejado.

O real padrão de saída obtido pela rede é então comparado com o padrão desejado. Dessa comparação surge um parâmetro de ajuste que será utilizado para calcular as correções dos pesos presentes na rede neural.

No aprendizado supervisionado diz-se ter a presença de um "professor" que observando os valores de saída produzidos pela rede "ensina" à mesma quais os valores de saída desejados.

Dentro do aprendizado supervisionado podemos considerar três tipos de paradigmas de aprendizado utilizados pelas RNAs.

Auto-associador - nesse paradigma treinamos a rede para armazenar em suas estruturas um conjunto de padrões onde todo padrão de entrada é igual ao seu respectivo padrão de saída. Terminada a aprendizagem, apresentamos à rede um desses padrões de entrada treinados só que de forma incompleta (partes do padrão) ou mesmo um padrão semelhante a um dos treinados, esperamos então que a rede consiga na presença desse padrão apresentado, recuperar o padrão original.
O processo que ocorre na rede é o de associação, onde o padrão apresentado é associado com um padrão original.

Associador de padrões - aqui um conjunto de pares de padrões é ensinado à rede, assim, espera-se que na presença de um dos membros do par, a rede consiga recuperar o outro membro do par. Procura-se com o associador de padrões um mecanismo no qual um conjunto arbitrário de padrões de entrada possa ser associado elemento a elemento com um conjunto arbitrário de padrões de saída.

Classificador - nesse paradigma existe um conjunto de padrões de entrada e categorias às quais os padrões de entrada devem ser classificados, isto é, todo padrão de entrada deve pertencer a alguma categoria. Treinamos então a rede de forma a ela saber classificar todo padrão de entrada na categoria desejada.
O objetivo desse paradigma é apresentar um dos padrões treinados ou um padrão ligeiramente destorcido de um padrão treinado à rede e obter a categoria a qual ele pertence, isto é, a rede saber classificar corretamente este padrão apresentado.

Uma variação do aprendizado supervisionado é o chamado aprendizado por reforço onde ao invés de termos um "professor" que diz à rede qual o padrão de saída desejado para cada padrão de entrada, teremos a figura de um "crítico".

Esse crítico, que denominamos **sinal de reforço**, é um sinal externo à rede e tem como papel indicar à mesma se ela teve um desempenho bom ou ruim com os seus padrões de saída.

De posse desse sinal que pode indicar recompensa ou punição, a rede altera os seus pesos de forma a melhorar seu desempenho e assim obter uma "crítica" mais positiva.

III.3.2 - Aprendizado não supervisionado.

Essa forma de aprendizado não requer nem os padrões de saída desejados e nem o sinal de reforço.

Como a rede não possui nenhuma forma de orientação (supervisão) exterior a respeito de seu desempenho, o aprendizado da rede não visa tentar satisfazer um desempenho previamente definido pelo usuário como no aprendizado supervisionado.

O objetivo desta forma de aprendizado é organizar as informações de entrada da rede de acordo com algumas características presentes nessas informações.

Cabe à rede desenvolver uma representação interna própria de forma a classificar padrões de entrada em conjuntos disjuntos tal que todos os elementos pertencentes a um conjunto sejam similares entre si, sendo que o critério de similaridade depende do algoritmo utilizado.

A classificação feita dos padrões de entrada depende além do algoritmo, também das características dos padrões, dos parâmetros de ajuste do algoritmo e até da ordem de apresentação desses padrões à rede.

Uma característica importante do aprendizado não supervisionado é que como ele desenvolve sua própria classificação de padrões, o resultado obtido pela rede pode ter ou não significado para o usuário da mesma.

O aprendizado competitivo [Die90] é a técnica mais representativa do aprendizado não supervisionado, nele as unidades de uma mesma camada competem entre si para ver qual delas obtém o direito de responder a determinados estímulos. A unidade vencedora tem seus pesos aumentados enquanto que as outras podem ter seus pesos diminuídos.

Como a rede é composta por unidades binárias (saída é 0 ou 1), a competição entre as mesmas se faz da seguinte forma: dentre todas as unidades de uma certa camada, somente aquela que possuir maior estímulo terá valor de saída 1, as outras terão valor de saída 0, daí o termo competição. Devido a esta característica as unidades são conhecidas como WTA ("winner-take-all").

Esse processo se repete ao longo da rede até chegar à camada de saída onde cada unidade representa uma classe. Como cada unidade adquiriu o direito de responder somente para alguns estímulos, podemos pensar que tais estímulos estão categorizados pela classe representada pela unidade.

Outras formas de classificação dos algoritmos de aprendizagem podem ser encontrados na literatura [Die90] como por exemplo: Mudanças Topológicas, Aprendizado Estocástico & Determinístico e Requerimentos de Entrada.

III.4 - O Algoritmo de Retropropagação do Erro - "Backpropagation".

Nesta parte descreveremos o algoritmo de aprendizagem supervisionado mais conhecido para perceptrons multicamadas, cuja idéia básica é retropropagar o erro obtido entre os valores de saída desejados e obtidos pela rede através das camadas da mesma, permitindo um ajuste de pesos que minimize tal erro.

Faremos uma introdução ao algoritmo seguida por uma explicação precisa de seu funcionamento, mostrando também algumas características e aplicações do mesmo. No final desta dissertação acrescentamos um apêndice (Apêndice I) que visa recordar ou esclarecer os conceitos matemáticos utilizados na descrição do algoritmo.

III.4.1 - Considerações Iniciais.

Em uma rede neural artificial a representação e armazenamento do conhecimento são feitos através dos pesos presentes nas ligações entre seus elementos de processamento (EPs).

Vários tipos de aprendizado conexionista foram utilizados para o armazenamento de conhecimento em redes associativas simples formadas por um conjunto de EPs de entrada diretamente conectadas a um conjunto de EPs de saída. Como essas redes não contém EPs "ocultos", isto é, EPs pertencentes a camadas ocultas, não há o problema de decidir o que esses EPs devem representar [Hin90]. Nestes casos as unidades da rede realizam uma classificação linear.

Na maioria das tarefas, porém, o mapeamento de vetores de entrada em vetores de saída possui uma estrutura complicada que só pode ser expressa utilizando-se uma rede com múltiplas camadas ocultas, tais redes, porém, sofrem do chamado problema de atribuição de crédito, que é o problema de como atribuir valores corretos aos pesos de forma que a rede tome as decisões corretas.

Dentre as várias soluções propostas para esse problema, uma das mais conhecidas é o procedimento de retropropagação do erro ("Backpropagation") que foi proposto em meados de 80 por Rumelhart, Hinton e Williams do grupo PDP [RHW86][RHW86a] com variantes desse procedimento descobertas independentemente por Parker [Par85], Werbos [Wer74] e Le Cun [Cun85].

Duas são as razões para a popularidade do algoritmo: a primeira é que o mesmo soluciona o problema de atribuição de crédito tornando teoricamente um perceptron multicamada capaz de aprender quaisquer classificações não lineares.

A segunda razão é o fato de que o mecanismo responsável pela operação do algoritmo é razoavelmente bem entendido, assegurando que a diferença entre as saídas desejadas e obtidas da rede, para todos os estímulos de treinamento, seja minimizada pelo cálculo repetido das derivadas parciais dessa diferença em relação aos pesos e modificando-se esses pesos em proporção a essas derivadas [Ken90].

O algoritmo de "Backpropagation" é um algoritmo supervisionado utilizado em redes alimentadas para frente ("feedforward networks"), isto é, não existem interconexões entre a saída de um EP e a entrada de outro EP na mesma camada ou em uma camada anterior, sendo que sua execução segue a seguinte idéia:

- Inicializam-se os valores dos pesos das ligações e limiares dos EPs com valores aleatórios,
- Lêem-se os valores de entrada e de saída desejados,
- Calcula-se a saída da rede obtida pelos cálculos feitos através de suas camadas,
- Calcula-se o erro entre a saída desejada e obtida,
- Mudam-se os pesos através de ajustes feitos caminhando-se na direção da camada de saída para a camada de entrada,
- Repete-se o processo até que a saída obtida da rede coincida com a saída desejada.

Atualmente existem várias versões do algoritmo de "Backpropagation", dentre as quais podemos citar a versão para aprendizado com reforço, a versão para aprendizado auto-supervisionado, "Backpropagation" para redes iterativas e também como um procedimento de probabilidade máxima [Hin90].

Limitaremos, porém, o nosso estudo à versão original do algoritmo de "Backpropagation", isto é, àquela com aprendizado supervisionado e em redes alimentadas para frente, por ser a forma mais conhecida e também pelo fato de suas idéias fundamentais estarem contidas nas outras formas variantes.

III.4.2 - Descrição do Algoritmo.

Como já dissemos, uma maneira pela qual podemos medir o desempenho de uma rede neural com o seu conjunto de pesos atuais pode ser feita pela comparação dos valores de saída obtidos com os valores de saída desejados para a rede.

Através dessa comparação podemos medir qual é o erro, denominado medida do erro, que a atual configuração de pesos está impondo à rede.

A medida do erro pode ser expressa por:

$$E = \frac{1}{2}\sum_{j,c}\left(y_{j,c} - d_{j,c}\right)^2 \qquad (6)$$

Onde:

$y_{j,c}$ é o estado atual da unidade de saída j no caso (exemplo) c e

$d_{j,c}$ é o estado desejado.

Como o estímulo que chega a uma unidade j, também chamado de "total de entrada" $\left(x_j\right)$ da unidade j, é usualmente uma função linear dos níveis de atividade das unidades que fornecem entradas para j, podemos representá-lo por:

$$x_j = -\theta_j + \sum_i y_i w_{ij} \qquad (7)$$

Onde:

y_i é o estado da i-ésima unidade,

$w_{i,j}$ é o peso da conexão da i-ésima para a j-ésima unidade e

θ_j é o limiar da j-ésima unidade.

E como o estado da j-ésima unidade $\left(y_j\right)$ é tipicamente definido como uma função não linear do seu total de entrada $\left(x_j\right)$, ou seja:

$$y_j = f\left(x_j\right) \qquad (8)$$

De (6),(7) e (8) podemos ver que a medida do erro é uma função que depende dos valores de entrada $(x_{entrada})$, dos limiares (θ), dos pesos das ligações (w)e dos valores de saída desejados (d), isto é,

$$E = E(x_{entrada}, \theta, w, d) \tag{9}$$

O objetivo de todo algoritmo de aprendizagem supervisionado é tornar seus valores de saída da rede $(y_{saída})$ tão próximos quanto possível dos valores de saída desejados (d), ou seja, minimizar o valor da função medida do erro (6).

Embora a medida do erro (6) seja função dos valores de entrada $(x_{entrada})$ e dos valores de saída desejados (d) ,temos que esses valores são fornecidos à rede são constantes e portanto, não podem ser alterados.

Pode-se considerar que um nó em uma rede neural que possua um limiar diferente de 0 é computacionalmente equivalente a um nó com um limiar igual a 0, tendo uma ligação extra conectada com um nó fictício com valor de saída sempre igual a -1 e com peso igual ao valor do limiar [Win92].

Como podemos considerar os limiares como pesos de ligações extras então podemos simplificar (9) para

$$E = E(w) \tag{10}$$

tornando o processo de minimizar a medida do erro como somente um ajuste dos valores dos pesos da rede.

O algoritmo de "Backpropagation" utiliza o método do gradiente descendente para minimizar a medida do erro, começando com um conjunto qualquer de pesos e repetidamente mudando cada peso por uma quantidade proporcional a $\partial E / \partial w_{ij}$ [Hin90]:

$$\Delta w_{i,j} = -\varepsilon \frac{\partial E}{\partial w_{ij}} \tag{13}$$

Quando ε, denominado tamanho do passo, tende a zero o número de atualizações tende a infinito, garantindo que o procedimento de aprendizagem convirja a um conjunto de pesos que corresponda a um mínimo global ou local para (6).

A medida do erro (6) é calculada como o somatório das medidas do erro para os vários casos de dados de entrada.

$$
\begin{aligned}
E &= \frac{1}{2}\sum_{j,c}\left(y_{j,c}-d_{j,c}\right)^2 \\
&= \sum_{c}\frac{1}{2}\sum_{j}\left(y_{j,c}-d_{j,c}\right)^2 \\
&= \sum_{c} E_c
\end{aligned}
\tag{12}
$$

onde

$$
E_c = \frac{1}{2}\sum_{j}\left(y_{j,c}-d_{j,c}\right)^2 \tag{13}
$$

De (12) podemos ver que a derivada parcial da medida do erro (6) com respeito a um particular peso pode ser calculada fazendo-se a soma da derivada parcial da medida do erro para cada caso de entrada [Win92].

$$
\frac{\partial E_c}{\partial w_{i,j}} = \frac{\partial E_c}{\partial y_j}\frac{dy_j}{dx_j}\frac{\partial x_j}{\partial w_{i,j}} \tag{14}
$$

Podemos então reduzir o excesso notacional do índice c, nos fixando em cada caso de entrada por vez, pois sabemos que cada peso será ajustado pela soma dos ajustes obtidos em cada caso.

O valor de $\partial E_c / \partial w_{i,j}$ é obtido pela regra da cadeia, de (7), (8) e (13)

$$
\frac{\partial E_c}{\partial w_{i,j}} = \frac{\partial E_c}{\partial y_j}\frac{dy_j}{dx_j}\frac{\partial x_j}{\partial w_{i,j}} \tag{15}
$$

considerando-se valores particulares para i e para j temos que de (7)

$$
\begin{aligned}
\frac{\partial x_j}{\partial w_{i,j}} &= \frac{\partial}{\partial w_{i,j}}\left[-\theta_j + \left(y_1 w_{1,j} + y_2 w_{2,j} + \ldots + y_i w_{i,j} + \ldots + y_n w_{n,j}\right)\right] \\
&= y_i
\end{aligned}
\tag{16}
$$

onde y_i pode ser a constante -1 caso se trate do nó fictício do limiar.

De (15) e (16) temos

$$
\frac{\partial E_c}{\partial w_{i,j}} = \frac{\partial E_c}{\partial y_j}\frac{dy_j}{dx_j} y_i \tag{17}
$$

A derivada parcial $\partial E_c/\partial y_j$ é fácil de ser calculada para a camada de saida. De (13) para um valor particular de j temos

$$\frac{\partial E_c}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\left((y_1-d_1)^2+(y_2-d_2)^2+\ldots+(y_j-d_j)^2+\ldots+(y_n-d_n)^2\right)\right] = (y_j-d_j) \tag{18}$$

Agora, para as camadas ocultas o cálculo é mais complicado. A idéia central do algoritmo de "Backpropagation" é que essas derivadas podem ser calculadas eficientemente começando-se com a camada de saída e calculando-se os valores das camadas anteriores com o uso dos valores das camadas já calculadas.

Para cada caso de entrada c, usa-se primeiramente o passo denominado "para frente" começando com as unidades de entrada e computando-se os níveis de atividade de todas as unidades da rede. Usa-se então o passo denominado "para trás" a fim de computar $\partial E/\partial y_j$ para todas as unidades ocultas, começando-se das unidades da camada de saída da rede.

Para cada unidade oculta j, pertencente à *j-ésima camada,* o único modo que ela pode afetar o erro da rede é via seus efeitos produzidos nas saídas das unidades k, na camada consecutiva k [Hin90].

Tem-se então:

$$\frac{\partial E_c}{\partial y_j} = \sum_k \frac{\partial E_c}{\partial y_k}\frac{\partial y_k}{\partial y_j} = \sum_k \frac{\partial E_c}{\partial y_k}\frac{dy_k}{dx_k}\frac{\partial x_k}{\partial y_j} \tag{19}$$

de (7)

$$\frac{\partial x_k}{\partial y_j} = \frac{\partial}{\partial y_j}\left(-\theta_k + \sum_j y_j w_{j,k}\right) = w_{j,k} \tag{20}$$

e de (19) e (20)

$$\frac{\partial E_c}{\partial y_j} = \sum_k \frac{\partial E_c}{\partial y_k}\frac{dy_k}{dx_k} w_{j,k} \tag{23}$$

Com relação à função não linear (8) que define o estado de uma unidade *j*, no algoritmo de "Backpropagation" que possui unidades com estados contínuos, uma típica função não-linear é a função sigmóide também conhecida como função logística [Hin90].

$$y_j = \frac{1}{1+e^{-x_j}} \tag{22}$$

A função sigmóide substitui a função-escada, que é mais fiel em relação à ação de um neurônio real, por apresentar quase o mesmo efeito da outra já que possuem formas semelhantes, além da vantagem matemática da suavidade da função que é essencial para o método do gradiente descendente.

Calculando dy_j/dx_j de (22) temos:

$$\begin{aligned}
\frac{dy_j}{dx_j} &= \frac{d}{dx_j}\left[\frac{1}{1+\mathrm{e}^{-x_j}}\right] \\
&= \frac{\mathrm{e}^{-x_j}}{\left(1+\mathrm{e}^{-x_j}\right)^2} \\
&= \frac{1}{\left(1+\mathrm{e}^{-x_j}\right)} - \frac{1}{\left(1+\mathrm{e}^{-x_j}\right)^2} \\
&= y_j - y_j^2 \\
&= y_j\left(1-y_j\right)
\end{aligned} \tag{25}$$

Juntando-se (18), (21) e (23) a (17) teremos:

$$\frac{\partial E_c}{\partial w_{i,j}} = \begin{cases} \left(y_j - d_j\right) y_j \left(1-y_j\right) y_i & \text{se } j \text{ é camada de saída} \\ \left(\sum_k \frac{\partial E_c}{\partial y_k} y_k \left(1-y_k\right) w_{j,k}\right) y_j \left(1-y_j\right) y_i & \text{se } j \text{ é camada oculta} \end{cases} \tag{26}$$

O algoritmo de "Backpropagation" é formado juntando-se (13), (14) e (26), tendo a seguinte forma:

⇒ Ajuste todos os pesos com valores aleatórios pequenos.

⇒ Escolha um tamanho do passo ε[1].

⇒ Até o desempenho ser satisfatório faça:

⇒ Para cada exemplo de entrada faça:

⇒ Compute a saída resultante da rede

⇒ Compute β para todos os nós da camada de saída z usando

$$\beta_z = y_z - d_z$$

⇒ Compute β para todos os nós da camada de saída j ($j \neq z$) onde

$$\beta_j = \sum_k w_{j,k} y_k (1 - y_k) \beta_k$$

⇒ Compute as mudanças do peso para todos os pesos usando

$$\Delta w_{i,j} = -\varepsilon y_i y_j (1 - y_j) \beta_j$$

⇒ Some todas as mudanças de peso para todos os exemplos de entrada adicionando a mudança final ao peso respectivo.

Lembrando sempre que os limiares são considerados pesos de ligações fictícias com valor de entrada sempre ajustada em -1.

III.4.3 - Características do algoritmo de "Backpropagation".

Dentre as possíveis características do algoritmo de "Backpropagation" duas são muito importantes por tratarem de questões fundamentais envolvendo algoritmos de aprendizagem: a garantia de convergência do algoritmo e a velocidade de convergência do mesmo.

A atuação do algoritmo de "Backpropagation" pode ser entendida através de uma interpretação geométrica, onde podemos tentar visualizar um espaço multidimensional que tenha um eixo para cada peso da rede neural e um eixo extra para a medida do erro (E) [Hin90].

Assim para cada combinação de pesos da rede neural existe um único ponto associado no eixo da medida do erro. Agindo assim podemos formar uma superfície n-dimensional chamada superfície do erro.

O algoritmo, ao alterar os pesos da rede, age como que caminhando nessa superfície de forma a encontrar o ponto onde ela tenha menor valor.

Em redes neurais sem camadas ocultas a superfície do erro tem somente um mínimo (mínimo global) caso exista uma solução perfeita (conjunto de pesos para a qual a medida do erro

[1]Ao invés de mover na direção determinada pelo Método do Gradiente Descendente até encontrarmos um mínimo relativo da função a ser minimizada, o algoritmo de "Backpropagation" opta por caminhar uma distância constante denominada tamanho do passo (ε).

tenha valor zero) e caso as unidades tenham funções de entrada e saída suaves e monotônicas. Nesse caso é garantido que o algoritmo de "Backpropagation" encontra a solução ótima.

Já nas redes com camadas ocultas, a superfície do erro pode apresentar além do mínimo global, alguns mínimos locais, mínimos estes que "enganariam" o algoritmo fazendo-o parar na sua busca pela solução ótima sem na realidade tê-la encontrado.

Na prática o problema do mínimo local não se apresenta como um problema sério pois para várias tarefas onde o algoritmo foi testado, a incidência das vezes onde ele ficou bloqueado em mínimos locais foi muito baixa [RHW86a].

A velocidade de convergência, talvez seja o maior problema do algoritmo de "Backpropagation", pois apesar de seu desempenho ser bastante impressionante em problemas relativamente pequenos, na prática ele se mostra, na sua atual forma, inadequado para tarefas maiores onde o seu tempo de aprendizado não tem bom escalonamento.

Em máquinas seriais o tempo de aprendizado é empiricamente da ordem de N^3, onde N é o número de pesos da rede [Hin90].

O tempo gasto com o cálculo dos valores da rede no sentido para frente (camada de entrada para a camada de saída) e no sentido para trás (camada de saída para a camada de entrada) é $O(N)$, o número de exemplos de treinamento é $O(N)$ e o número de vezes que os pesos precisam ser atualizados é aproximadamente $O(N)$.

Em máquinas paralelas usando-se um processador para cada conexão da rede o tempo de aprendizado pode ser reduzido a $O(N^2)$.

III.4.4 - Aplicações.

O algoritmo de "Backpropagation" por ser de simples entendimento e por apresentar desempenho satisfatório se tornou um dos algoritmos de aprendizagem mais populares atualmente. Devido à sua repercussão ele foi experimentado em um número enorme de aplicações dentre as quais podemos citar as mais clássicas.

O problema do XOR , *ou exclusivo* [CS88], é interessante pois sua solução requer a presença de unidades ocultas e também por ser um subproblema comum de problemas mais complexos.

O grupo de pesquisa do PDP [RHW86a] utilizou dois modelos de redes neurais sendo um com uma única unidade oculta e ligações diretas da camada de entrada para a camada de saída e outro com duas unidades ocultas e sem ligações diretas entre as camadas de entrada e de saída.

O sistema foi ensinado a solucionar o problema do XOR centenas de vezes e somente em dois casos o sistema encontrou um mínimo local, não encontrando, portanto, a solução do problema, esses dois casos envolveram a versão da rede com duas unidades ocultas.

Embora não sabendo a freqüência de ocorrência de tal mínimo local, a experiência do grupo com esse e outros problemas é de que esses mínimos são bastante raros.

O problema do T e C [CS88], é o problema geométrico de distinguir entre as letras escritas T e C independentemente da sua rotação e translação.

A arquitetura usada por Rumelhart et al [RHW86a] para solucionar o problema foi um perceptron com três camadas onde a primeira camada era denominada de campo receptivo e a última camada era composta de apenas uma unidade motora. O padrão de conexões entre as camadas procurava imitar a arquitetura da parte do cérebro responsável pela visão. O algoritmo de "Backpropagation" foi usado para ajustar os pesos das unidades ocultas, sendo necessárias de cinco a dez mil apresentações dos padrões T e C para que a rede pudesse classificar os padrões apresentados.

Uma outra aplicação interessante feita por Sejnowski e Rosenberg [SR87] é a **NETtalk** [Hin90] onde uma rede com uma camada oculta pode ser treinada para pronunciar letras surpreendentemente bem.

As unidades de entrada da rede codificam as letras a serem pronunciadas usando uma unidade diferente para cada letra. Elas codificam também o contexto onde a letra aparece através de uma janela que percorre o texto e que enxerga sete letras sendo a letra central aquela a ser pronunciada.

A camada do meio conecta a camada de entrada com a camada de saída e essa codifica o fonema a ser pronunciado usando 21 características articulatórias e 5 características para stress e limites silábicos.

Com um conjunto de mil palavras e após vinte mil apresentações a rede está pronta para ler e falar com precisão de 95%. A rede também generaliza bem com novos exemplos, demonstrando que ela captura as regularidades do mapeamento.

III.5 - Considerações Finais.

O assunto de aprendizagem em RNAs é bastante vasto e complexo pois as redes funcionam a nível de unidades e ligações e não a nível de conceitos, dificultando em muito a compreensão de como as mesmas mapeiam as informações de entrada em informações de saída.

Por não se saber como a rede está tomando as decisões, surge a dificuldade de encontrar uma forma de aprendizagem que seja adequada para um dado modelo de rede e que tenha o melhor desempenho para todas as aplicações nas quais o modelo propõe solução.

Waltz e Feldman [WF88] afirmam que até o momento todos os sistemas construídos ou aprenderam um número limitado de itens ou foram construídos para problemas específicos nunca apresentando um aprendizado absolutamente geral, isto é, aquele que funcionasse para qualquer tipo de problema apresentado.

Não se tendo um algoritmo que seja absolutamente geral, isto é, que funcione bem para todas as situações surgem muitas questões importantes para o tema de aprendizagem:

Qual dentre os inúmeros existentes que mais se adapta à rede escolhida para a solução do problema?

Qual a quantidade ideal de informações a ser fornecida à rede nessa fase de treinamento?

Fornecendo-se pouca informação à respeito do domínio do problema, a capacidade de generalização e mesmo de solução do problema estará sendo limitada, poder-se-ia fornecer então a

maior quantidade possível de informações à rede, mas aí o responsável pela mesma estará incorrendo em mais dois problemas.

O problema do treinamento excessivo, onde cada informação adicionada à rede pode comprometer as que já estão nela representadas fazendo com que a rede "desaprenda" as informações já armazenadas em sua estrutura.

Como cada ligação armazena "pedaços" de várias informações, isto é, o conhecimento está totalmente distribuído pela rede, não se pode prever como o acréscimo de uma informação nova na rede pode alterar as informações já armazenadas.

O segundo problema é que todo treinamento não somente exige um tempo proibitivo como também um grande número de exemplos sem garantias de convergência.

Para que a rede "aprenda" uma determinada informação, faz-se necessária a apresentação da mesma à rede, através de sua camada de entrada, um certo número de vezes. Se a quantidade de informações a ser aprendida for grande, o número de apresentações também o será, tornando, portanto, o tempo total de aprendizagem muito grande.

Quanto maior for o número de informações a serem aprendidas, menor a possibilidade da rede conseguir fazê-lo, isto é, o algoritmo de aprendizagem conseguir convergir para uma configuração de pesos que represente toda a informação armazenada.

Outra questão também em aberto é a maneira como as informações de entrada devem ser apresentadas à rede.

Uma RNA pode "aprender" suas informações de entrada de duas maneiras: em estágios ou simultaneamente. Na primeira forma cada informação é apresentada à rede até que ela seja assimilada pela mesma, só então se passa para a apresentação da informação seguinte.

Na outra forma, a apresentação simultânea, todas as informações são apresentadas uma vez à rede e depois repete-se o ciclo reapresentando todas as informações novamente, até que o algarismo convirja , isto é, a medida do erro ser a menor possível. Quando a rede "aprender" uma informação ela então terá "aprendido" todas, ao contrário da forma em estágios, onde ela "aprende" uma por vez.

O problema é que conforme o algoritmo de aprendizagem e as informações de entrada, essa seqüência de apresentação das informações (estágio ou simultaneamente) pode alterar significativamente o número de apresentações necessárias para o algoritmo convergir, sem que haja nenhuma informação a priori de qual a maneira mais adequada para cada caso.

Por todas estas e outras questões existentes, concluímos que sem um entendimento do funcionamento dos modelos de RNAs, isto é, sem abrir essa "caixa preta" que funciona em baixo nível, não se conseguirá encontrar a solução para todas as questões e problemas formulados para aplicações específicas quanto mais para o desenvolvimento de mecanismos gerais. E é na direção de abrir a "caixa preta" que foi implementado o sistema descrito neste trabalho.

[illegible]

Capítulo IV

Objetivo do Trabalho

Um dos grandes mistérios que ainda restam para a ciência moderna é o funcionamento do cérebro humano.

O conhecimento sobre o cérebro é tanto melhor quanto menor for o nível que estivermos observando, isto é, conhece-se bem a estrutura dos neurônios e o mecanismo de transmissão dos impulsos nervosos e razoavelmente bem as redes neurais especializadas existentes dentro do cérebro, mas muito mal o funcionamento de porções inteiras.

É desnecessário dizer que o perfeito entendimento do cérebro como um todo, isto é, o entendimento, dentre outras coisas, do funcionamento a nível biológico dos pensamentos, sentimentos, lembranças, etc, é ainda mera especulação.

Tentar entendê-lo é como tentar decifrar um algoritmo de milhões de linhas de código, rodando em uma máquina de processamento paralelo, somente através da observação dos zeros e uns presentes na memória. Podemos dizer que o problema chave do entendimento do funcionamento do cérebro está em seu baixo nível de funcionamento, o nível dos neurônios.

As Redes Neurais Artificiais (RNAs) como o próprio nome indica, surgiram da tentativa de modelar as redes biológicas, sendo, portanto, de se esperar que as mesmas apresentem dificuldades de entendimento semelhantes às existentes nas Redes Neurais Biológicas (RNBs).

Podemos entender o funcionamento dos elementos de processamento (EPs) e dos pesos das ligações isoladamente, mas encontramos dificuldades quando tentamos explicar como estas entidades agem coletivamente.

Deste não entendimento coletivo, surgem várias questões com respostas nem sempre triviais tais como:

- Como usar as entradas da rede para expressar o que se sabe?
- Como usar as saídas da rede para determinar o que se quer saber?
- Quantas unidades a rede deve ter?
- Como devem ser distribuídas as unidades dentro da rede?
- Como apresentar as informações de entrada, em estágios ou simultaneamente?
- Qual a quantidade ideal de informações que deve ser fornecida à rede em seu treinamento?

- Que tipo de informações devem ser oferecidas à rede?

Diferentemente do cérebro que tem um bom funcionamento mesmo sem termos um perfeito entendimento sobre seus mecanismos, as RNAs muitas vezes dependem deste bom entendimento para apresentarem um desempenho satisfatório.

Portanto, de uma forma ou de outra, as pessoas que se envolvem com as RNAs precisam de pelo menos uma indicação de respostas para estas e várias outras perguntas deixadas em aberto.

Como ainda não é claro e bem estabelecido o funcionamento das RNAs, não existem ainda boas e completas metodologias para a utilização das mesmas em aplicações, isto é, metodologias que diante de um problema específico a ser resolvido, indiquem qual a topologia de rede, o algoritmo de aprendizagem e a amostragem de informações adequadas ao funcionamento desejado.

Não queremos com isto dizer que as RNAs careçam de fundamento teórico, muito pelo contrário, mas sim que o mesmo não é suficiente para dar todas as respostas às questões envolvidas na solução de problemas com as RNAs e nem de fácil compreensão para permitir sua ampla utilização por parte de todos os usuários das redes.

Em não se tendo uma metodologia que indique a combinação ótima dos elementos de uma RNA para uma determinada aplicação, resta aos usuários das redes a opção de partir de sua base teórica e utilizando-se de métodos empíricos ir formando suas regras subjetivas e portanto, indicações pessoais de como conseguir as melhores combinações dos elementos das RNAs para as diversas situações de utilização das redes.

Esta técnica, porém, encontra muitas dificuldades em sua realização devido à grande quantidade de variáveis que precisam ser avaliadas durante todo o processo de desenvolvimento da rede.

Partindo-se de uma configuração inicial de escolhas podemos chegar à conclusão de que ela não é boa o suficiente, restando-nos então a tarefa de decidir qual ou quais elementos desta configuração devem ser alterados para a obtenção de um melhor resultado (vide figura IV.1).

Somente com uma boa familiarização com as variáveis envolvidas no desenvolvimento de uma RNA é que o usuário poderá ter indicações de quais são as melhores modificações a serem feitas na rede para a melhoria de seu desempenho.

Nesta busca pelo entendimento do funcionamento das RNAs através da familiarização do usuário com os seus elementos formadores (denominaremos este processo de criação de modelos mentais do funcionamento da RNA) é que se localiza o presente trabalho.

Com o fornecimento de ferramentas através de sua interface gráfica, o Sirena procura minimizar a dificuldade de entendimento dos processos de baixo nível realizados pelas RNAs facilitando, em caso de sucesso, o desenvolvimento e a utilização das mesmas.

O objetivo foi desenvolver uma ferramenta de simulação de RNAs, que durante o processo de simulação apresentasse representações alternativas das alterações mais relevantes na rede.

As representações deveriam ter componentes tanto qualitativos (cores indicando o aumento ou não da freqüência de disparo dos EPs) quanto quantitativos (relação numérica das alterações que ocorrem).

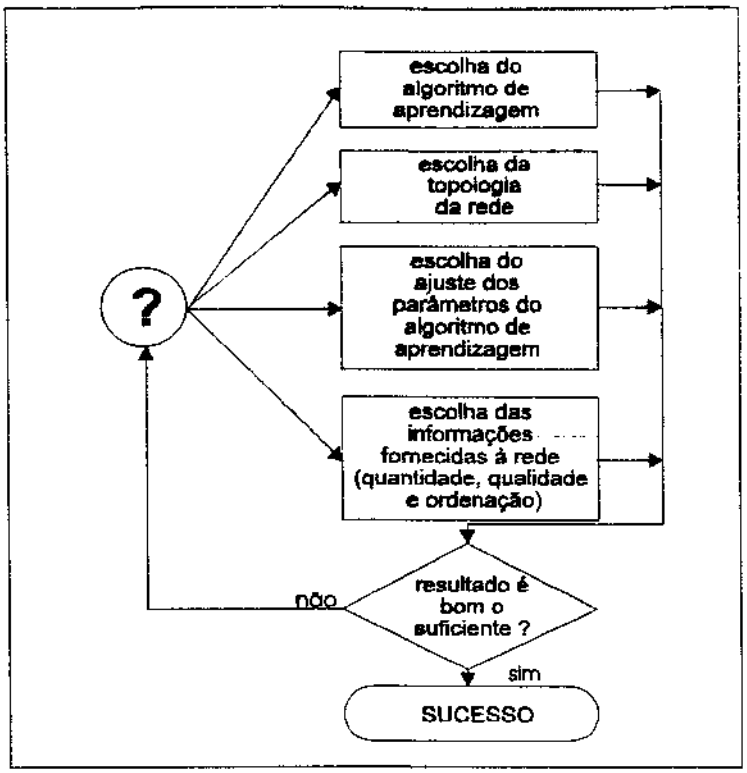


Figura IV.1 - Possível ciclo de desenvolvimento de uma RNA.

Além disso objetivamos a criação de uma interface de fácil manipulação e alto nível de interatividade. O usuário define a topologia, o algoritmo de aprendizagem, saídas, entradas, limiares, etc., e a qualquer momento interrompe o processo e efetua alterações em suas escolhas.

Portanto, nossas escolhas privilegiaram aspectos relacionados ao entendimento e não ao desempenho. Com isso, temos como objetivo os seguintes usuários alvo:

- o novato no campo das RNAs que utiliza a ferramenta para a verificação dos aspectos teóricos básicos do assunto;
- o projetista de uma rede neural para aplicações no mundo real que não tendo acesso ao conhecimento matemático envolvido na análise da mesma, utiliza o processo empírico para o desenvolvimento de suas aplicações.
- o estudioso que quer aprofundar seus conhecimentos sobre o funcionamento das RNAs.
- e finalmente o pesquisador teórico que utiliza a ferramenta como um passo intermediário antes da análise quantitativa de um modelo de rede neural.

No próximo capítulo faremos uma descrição completa da ferramenta construída.

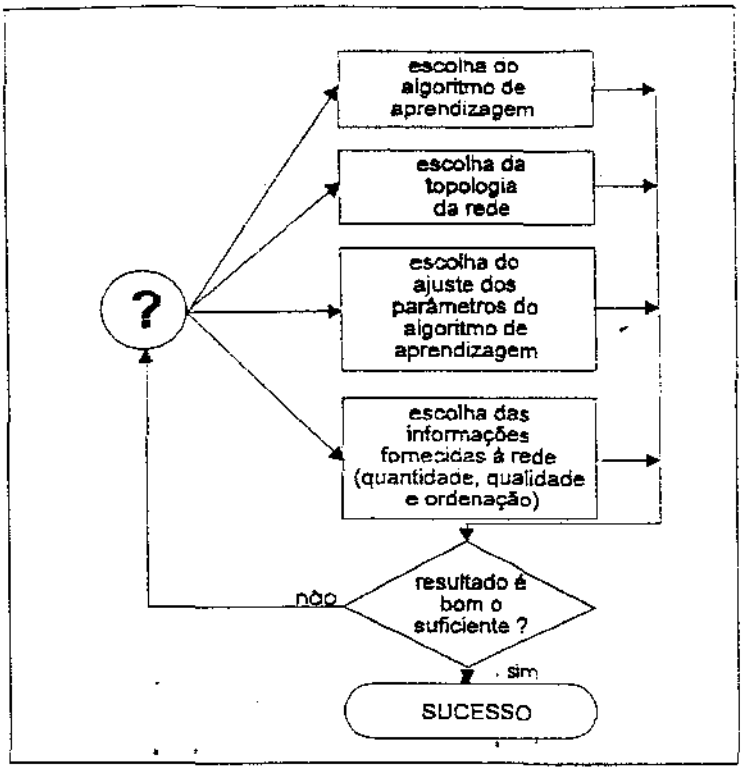


Figura IV.1 - Possível ciclo de desenvolvimento de uma RNA.

Além disso objetivamos a criação de uma interface de fácil manipulação e alto nível de interatividade. O usuário define a topologia, o algoritmo de aprendizagem, saídas, entradas, limiares, etc., e a qualquer momento interrompe o processo e efetua alterações em suas escolhas.

Portanto, nossas escolhas privilegiaram aspectos relacionados ao entendimento e não ao desempenho. Com isso, temos como objetivo os seguintes usuários alvo:

- o novato no campo das RNAs que utiliza a ferramenta para a verificação dos aspectos teóricos básicos do assunto;
- o projetista de uma rede neural para aplicações no mundo real que não tendo acesso ao conhecimento matemático envolvido na análise da mesma, utiliza o processo empírico para o desenvolvimento de suas aplicações.
- o estudioso que quer aprofundar seus conhecimentos sobre o funcionamento das RNAs.
- e finalmente o pesquisador teórico que utiliza a ferramenta como um passo intermediário antes da análise quantitativa de um modelo de rede neural.

No próximo capítulo faremos uma descrição completa da ferramenta construída.

Capítulo V

Descrição dos Recursos do Sirena e sua Utilização

Este capítulo descreve o sistema Sirena através de sua interface gráfica, apresentando os objetivos e as características de cada uma das suas ferramentas.

Procurou-se na maior parte do capítulo seguir um exemplo inicial de RNA objetivando facilitar o acompanhamento e o entendimento, por parte do leitor, do significado e da utilização dos recursos que compõem o Sirena.

V.1 - Aspectos Gerais.

O Sirena é um sistema que objetiva auxiliar no processo de entendimento do funcionamento dos modelos de RNAs, concebido de forma a favorecer uma grande interação com o usuário, um cruzamento das informações do sistema, a liberdade de manipulação dos recursos implementados e a possibilidade de uma visão qualitativa das informações fornecidas.

Para tanto, várias características foram introduzidas na ferramenta desenvolvida, dentre as quais podemos citar:

- *Visualização da rede* - permite o acompanhamento do funcionamento da rede através da visualização de um grafo que a representa, onde os valores de saída dos EPs vão sendo mostrados através de coloração e variação de tamanho dos nós.
  Objetiva-se com esses recursos uma visão de conjunto da rede, possibilitando o entendimento do funcionamento da mesma como um todo e não somente como a soma de suas partes.
- *Zoom* - possibilita a visualização do funcionamento de um pedaço da rede ou da rede inteira conforme a opção escolhida.
  Objetiva-se aqui, ao contrário do item anterior, o entendimento isolado de uma parte constituidora da rede.
- *Informação sobre a situação da rede* - mostra informações sobre a rede com número total de EPs, número de EPs por camada, além do número atual de camadas.

Objetiva-se desenvolver a sensibilidade sobre o relacionamento entre a quantidade de EPs e sua distribuição em camadas com o desempenho da rede.

- *Armazenamento total de informações* - mecanismo cuja função é permitir que todas as informações que o simulador é capaz de manipular tais como os exemplos de aprendizado, os valores dos pesos das ligações e os resultados da rede, sejam armazenadas para futura verificação e possível comparação com outras simulações. O ativamento deste mecanismo fatalmente implica em uma diminuição da velocidade.
  Objetiva-se mostrar o relacionamento de um estado atual da rede com um estado da mesma em situação anterior ou a comparação do estado de uma outra rede com o estado atual da rede em processamento.
- *Visualização da situação da rede* - mostra analiticamente todas as informações relacionadas à rede durante as fases de ajuste e inferência, tais como valores de saída dos EPs, valores de pesos e de limiares.
  Objetiva-se caracterizar com precisão o estado atual de todos os elementos formadores da RNA através de suas respectivas grandezas numéricas.
- *Entrada e saída da rede via editor apropriado* - os dados de entrada e de saída da rede podem ser introduzidos usando um editor apropriado nas fases de aprendizagem e/ou inferência.
  Tais dados podem ser armazenados, juntamente com as outras informações do simulador durante esses dois processos e visualizados durante a sua utilização.
  Objetiva-se além de facilitar de introdução dos dados de entrada e de saída, a comparação e verificação de evolução entre formatos geométricos, das informações de entrada e saída fornecidas e das informações devolvidas pela rede no editor apropriado.
- *Criação e alteração de topologia de uma RNA Padrão*[1] - desde a topologia mais simples para o entendimento básico até as mais complexas permite-se a:

  ⇒ Criação dos elementos de processamento (EPs) com suas respectivas interconexões (ligações).

  ⇒ Eliminação de EPs e/ou ligações existentes.

  ⇒ Alteração dos valores limiares das unidades.

  ⇒ Alteração dos valores dos pesos das ligações.

  Objetiva-se a liberdade de criação/modificação da rede como forma de auxiliar o entendimento da relação entre a topologia e os outros componentes de processamento.
- *Modificação dos parâmetros de ajuste do algoritmo de "Backpropagation"* - possibilita a modificação de parâmetros de ajuste do algoritmo, os termos de ganho e de momento, que alteram o seu desempenho.

[1]Denominaremos de *RNA Padrão* toda rede alimentada para frente cujas ligações unam somente EPs de camadas consecutivas, isto é, se uma ligação une um EP da camada i com outro pertencente à camada j então devemos ter a relação $j = i + 1$.

Objetiva-se a verificação da variação do desempenho de aprendizagem da rede conforme a topologia da rede e as informações a ela fornecidas.

- *Modo de execução* - permite que o usuário controle o processo de simulação passo-a-passo ou por etapas.
  No processo "passo-a-passo" pode-se verificar a contribuição que cada iteração do algoritmo de aprendizagem produz na rede, enquanto que no processo "por etapas" pode-se verificar a evolução da aprendizagem da rede quando várias iterações do algoritmo são executadas.

- *Retroexecução* - tendo todos os dados da simulação armazenados, o usuário poderá retroceder na simulação um certo número de passos, de forma a poder parar a execução em uma iteração anterior de seu interesse e continuar sua execução de forma normal, isto é, para frente, com exemplos de entrada diferentes ou com a topologia alterada.
  Objetiva-se permitir as usuário um maior controle sobre o processamento, onde o usuário não satisfeito com a forma de evolução do desempenho da rede, tem a possibilidade de mudar a direção dessa evolução em algum uma etapa anterior da aprendizagem.

- *Desativação do histórico* - opção por desligar a modalidade de histórico.
  Objetiva-se aumentar a velocidade de simulação, caso o usuário não tenha interesse nos dados a serem armazenados.

- *Alteração da topologia em tempo de execução* - facilidade útil no desenvolvimento de novas topologias ou análise de pequenas alterações na topologia corrente.
  Objetiva-se a análise, a qualquer memento da execução, dos efeitos da topologia no desempenho da rede.

- *Alteração da variação de cor e tamanho dos EPs da rede* - permite que o usuário escolha, que na representação da rede através do grafo, os nós representando os EPs variem ou não de cor e/ou tamanho conforme o valor de saída dos mesmos.
  Objetiva-se um entendimento qualitativo do funcionamento da RNA, isto é, a verificação da mudança de estado da rede se dá pela observação das cores e tamanhos e não valores numéricos que descrevem este estado.

- *Utilização de vários arquivos para o armazenamento de informações* - pode-se optar pela gravação dos processos de simulação e inferência em um ou mais arquivos externos.
  Objetiva-se a possibilidade de comparação de informações entre vários modelos de redes ou de várias processos de inferências/aprendizagem em um mesmo modelo de rede.

Todas as ferramentas do Sirena podem ser acessadas pelo usuário através de sua interface gráfica constituída de várias janelas do ambiente Windows, sendo que cada uma das janelas implementa uma ou mais das características acima descritas.

Uma apresentação típica destas janelas pode ser vista na figura V.1.

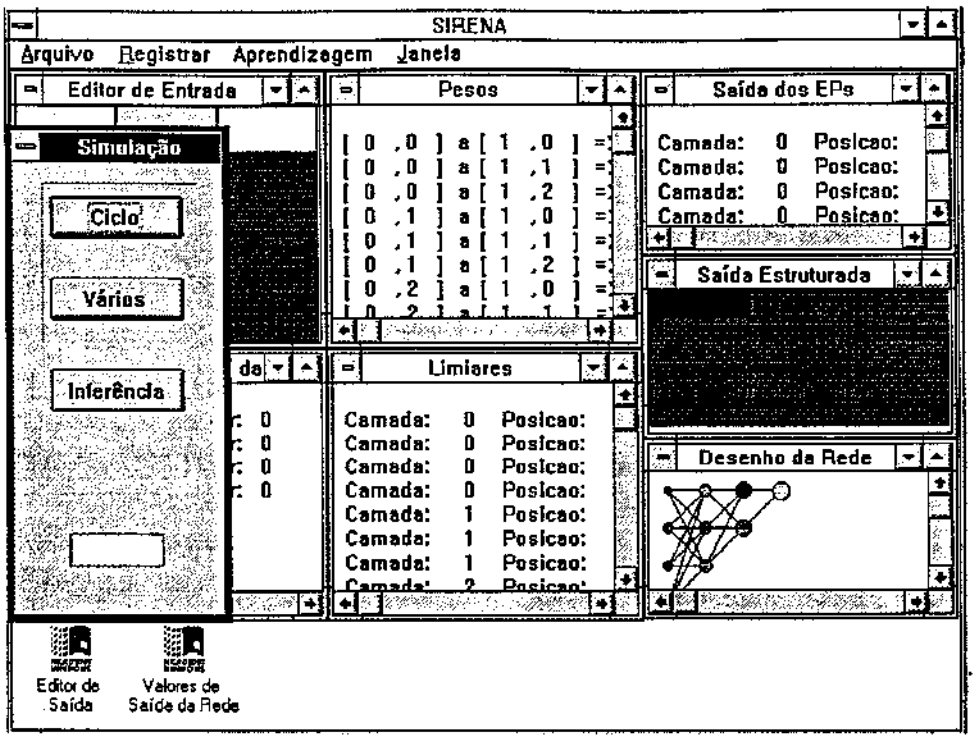


Figura V.1 - Tela exemplo do sistema Sirena.

Pelo fato da ferramenta estar implementada usando a Interface de Múltiplos Documentos (MDI), suas janelas apresentam aspectos semelhantes tais como barras de rolagem, barra de título, moldura, ícone do sistema que permite as opções de restaurar, mover, fechar, minimizar e maximizar a janela e ícones de maximização e minimização [Pet93].

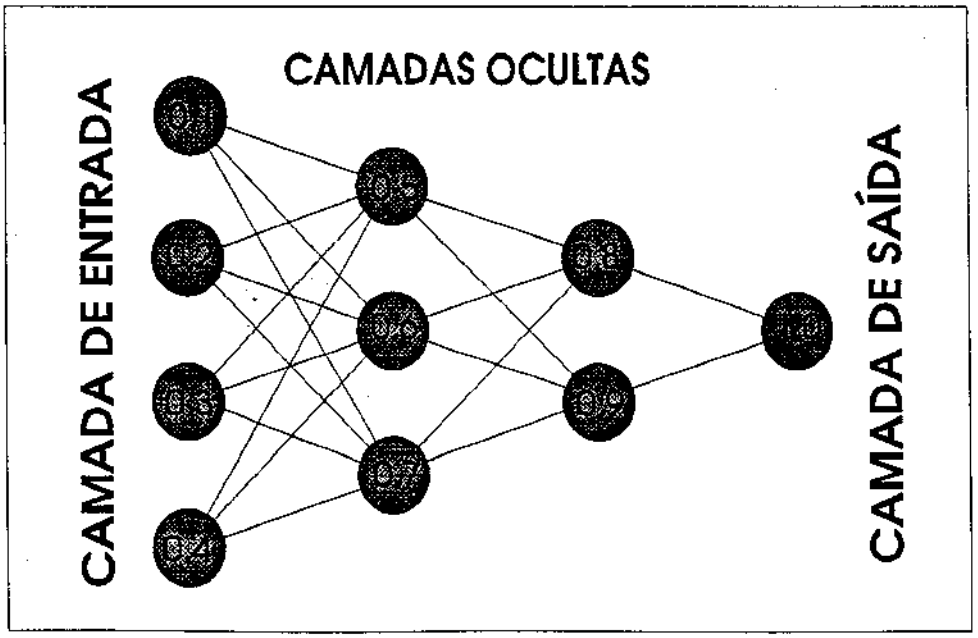


Figura V.2 - Grafo da RNA Exemplo.

A maior janela da ferramenta, aqui denominada Janela Principal do Sirena, apresenta em sua barra de título o nome SIRENA e tem como funções: criar, destruir e organizar dentro de si as demais janelas.

As demais janelas da ferramenta se dividem em dois grupos de acordo com a forma que apresentam as suas informações: as janelas textuais e as janelas gráficas. Para que possamos entendê-las melhor, vamos supor um modelo exemplo de RNA implementado no simulador, a fim de que todas as janelas a serem apresentadas durante a descrição do sistema Sirena apresentem informações coerentes entre si.

Esse modelo exemplo, que nos referiremos daqui por diante como **RNA Exemplo**, é formado por quatro camadas compostas de quatro EPs na camada de entrada, três EPs na segunda camada, dois EPs na terceira e finalmente um EP na camada de saída, tendo seus respectivos valores de saída mostrados na figura V.2.

V.2 - JANELA PRINCIPAL.

A Janela Principal do Sirena é assim denominada por ser a maior das janelas da ferramenta. É através dela que são criadas as demais janelas da ferramenta e é em seu interior que as demais estão dispostas.

A Janela Principal quando não possuir nenhuma janela ativa dentro de si, apresenta somente um *"menu"* contendo um único item, o item "Arquivo".

Neste item podemos abrir um *"submenu"* dividido em três grupos como pode ser visto na figura V.3.

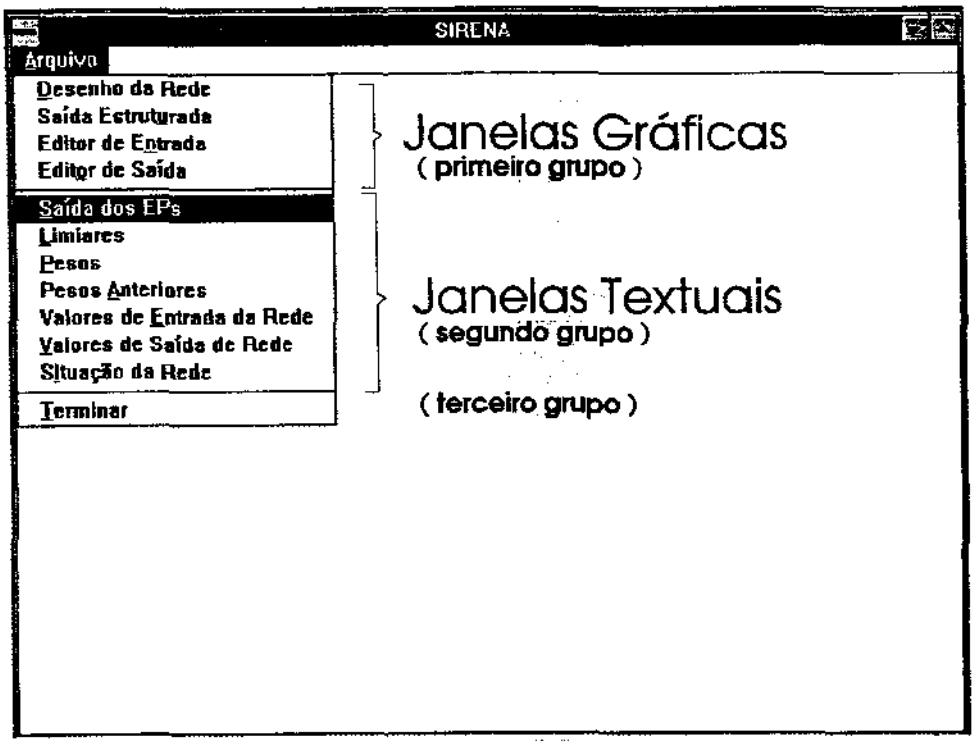


Figura V.3 - *"Menu"* da Janela Principal.

O primeiro grupo permite a abertura das quatro janelas gráficas da ferramenta, o segundo, a abertura das sete janelas textuais e o terceiro e último, permite encerrar a atividade da ferramenta, fechando a Janela Principal e todas as janelas gráficas e textuais que estiverem abertas.

Quando pelo menos um janela estiver sendo apresentada, esteja ela aberta ou mimizada, o *"menu"* da Janela Principal tem o seu aspecto alterado em relação ao número de itens apresentados. Na figura V.4 tem-se a Janela Saída dos EPs minimizada e a respectiva alteração do *"menu"*.

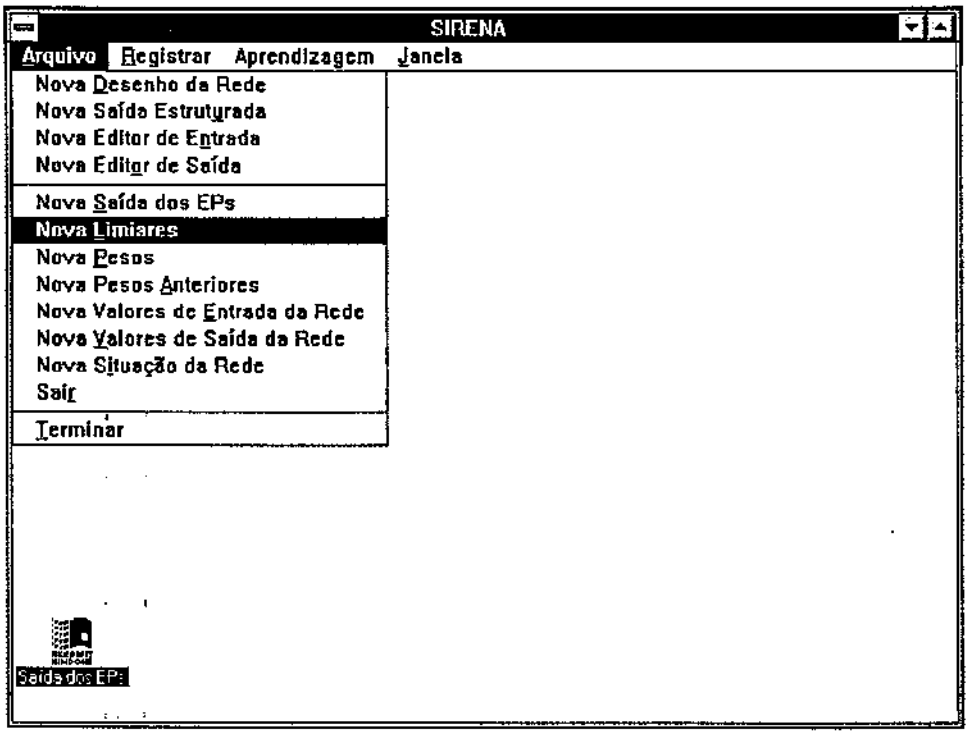


Figura V.4 - *"Menu"* das janelas textuais e gráficas.

O *"menu"* agora apresenta três itens adicionais em relação ao anterior sendo que o item Arquivo deste novo *"menu"* é semelhante ao do *"menu"* da Janela Principal mostrado anteriormente, onde as únicas diferenças são: a palavra "Nova" antes dos nomes das janelas indicando a opção de criação de uma nova janela da ferramenta e o item "Sair" que permite encerrar a janela atualmente ativa.

Passaremos a seguir à descrição das diversas janelas da ferramenta criadas através do *"submenu"* do item Arquivo.

Os demais itens do *"menu"* principal serão gradativamente apresentadas à medida que forem sendo descritas as características da ferramenta.

V.3 - JANELAS TEXTUAIS.

Pertencem a essa classificação as janelas: Saída dos EPS, Limiares, Pesos, Pesos Anteriores, Valores de Entrada da Rede, Valores de Saída da Rede e Situação da Rede.

São denominadas de textuais por apresentarem apenas informações escritas em seu interior como mostrado na figura V.5.

Como estas janelas objetivam apenas mostrar informações relacionadas com o estado atual do simulador, são, portanto, as que menos oferecem possibilidade de iteração com o usuário.

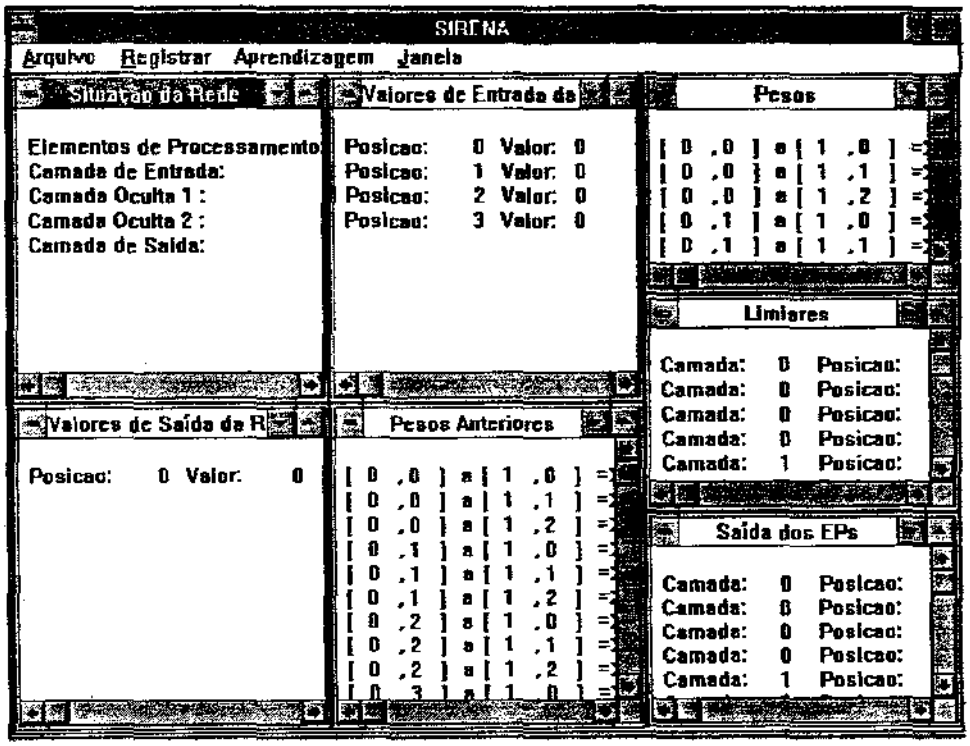
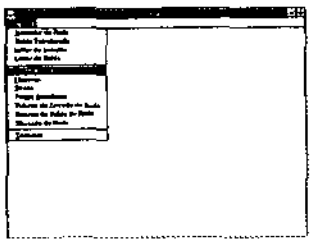


Figura V.5 - Janelas textuais.

A seguir vamos descrever cada uma das sete janelas com mais detalhes.



V.3.1 - Janela Saída dos EPs.

Esta janela, que é a primeira das janelas textuais, está mostrada na figura V.6 em sua forma maximizada.

Ela apresenta os valores de saída da função de transferência dos EPs da rede. Para cada EP ela mostra a camada a que ele pertence, sua posição dentro da camada e finalmente o valor de sua função de transferência. Para o algoritmo de "Backpropagation" a função de transferência utilizada é a função sigmóide.

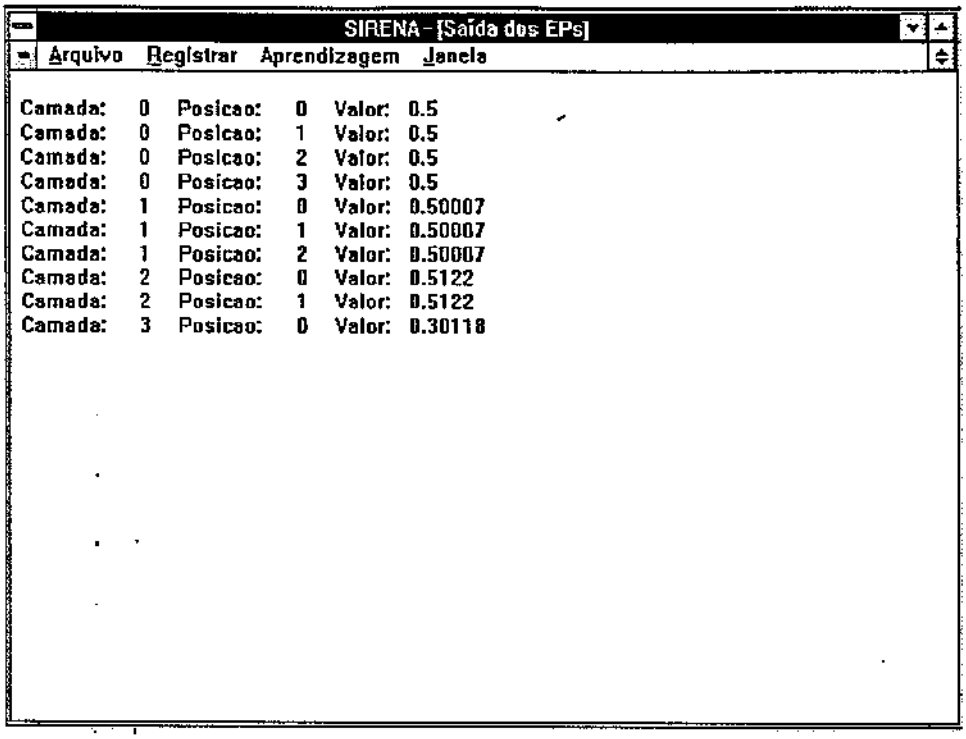
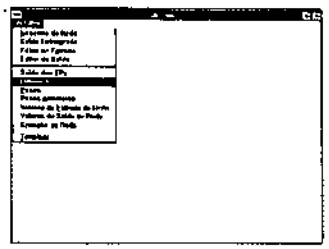


Figura V.6 - Janela Saída dos EPs.

A camada de entrada da rede sempre é considerada como a camada zero e o primeiro EP dentro de cada camada sempre ocupa a posição zero.



V.3.2 - Janela Limiares.

Esta janela, como o próprio nome diz, apresenta o valor do limiar correspondente a cada EP da rede.

Sua forma de apresentação é idêntica à da Janela de Saída dos EPs, variando somente na quantidade numérica apresentada pelo campo Valor.

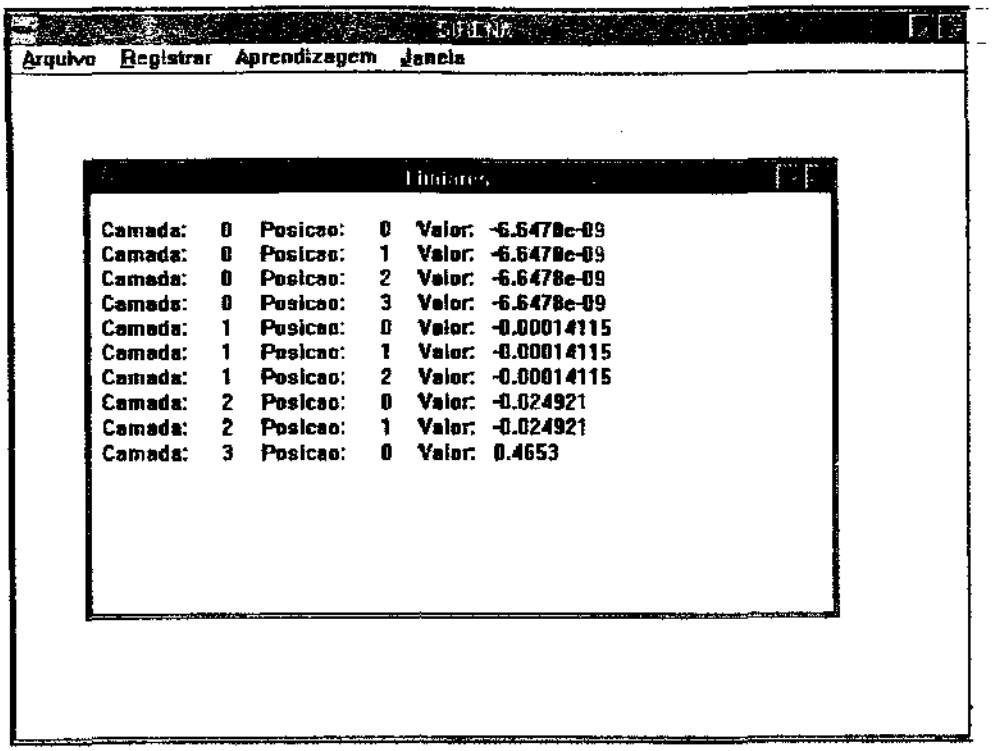
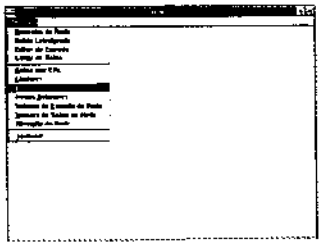


Figura V.7 - Janela Limiares.



V.3.3 - Janela Pesos.

Apresenta, como mostrado na figura V.8, o valor dos pesos das ligações que unem os EPs formadores da rede simulada.

Sua apresentação segue o seguinte formato:

[camada do EP inicial , posição dentro da camada] **a** [camada do EP final , posição dentro da camada] => **Valor** : valor do peso da ligação.

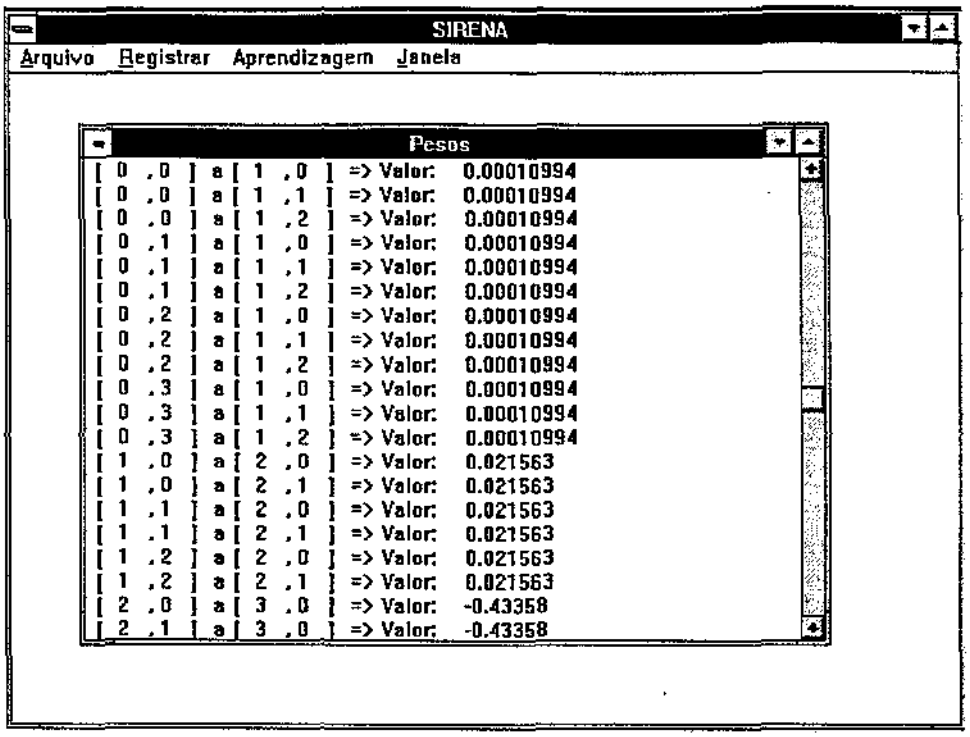
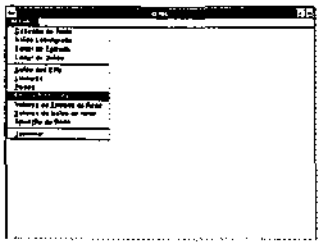


Figura V.8 - Janela Pesos.



V.3.4 - Janela Pesos Anteriores.

Semelhante à Janela Pesos, ela também mostra os valores dos pesos das ligações, só que os da iteração anterior do algoritmo de aprendizagem.

Antes que a primeira iteração do algoritmo de aprendizagem ocorra, as duas janelas apresentam valores iguais, isto é, os valores dos pesos são iguais aos dos pesos anteriores. Depois da primeira iteração esta janela apresenta os valores dos pesos da iteração anterior, isto é, se a Janela Pesos estiver mostrando os pesos da rede ajustados na *i*-ésima iteração do algoritmo de aprendizagem, a Janela Pesos Anteriores estará mostrando os pesos ajustados na (*i*-1) ésima iteração.

Seu formato de apresentação segue o modelo da Janela Pesos como pode ser visto na figura V.9.

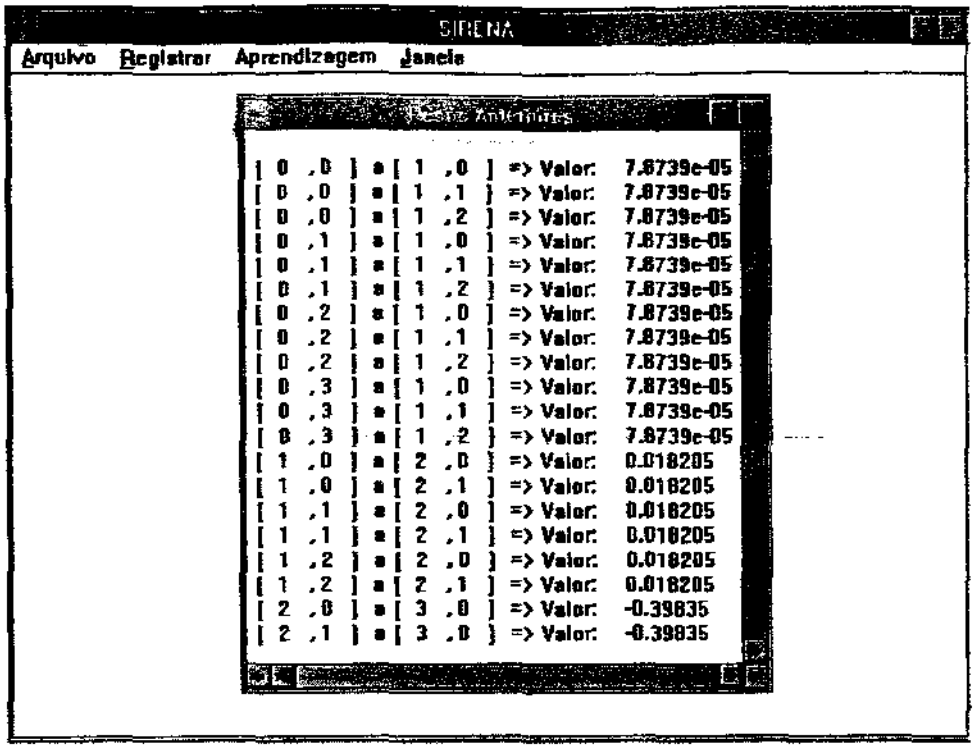
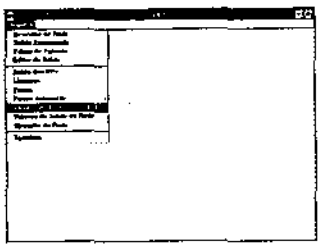


Figura V.9 - Janela Pesos Anteriores.



V.3.5 - Janela Valores de Entrada da Rede.

A Janela Valores de Entrada da Rede, representa o vetor de entrada da rede, isto é, aquele que fornece as informações do mundo externo para a camada de entrada da rede. Neste vetor cada uma das suas entradas está associada a um EP da camada de entrada.

Esta janela apresenta apenas dois valores: a posição dentro do vetor e o valor de entrada para aquela posição. A figura V.10 mostra que os EPs zero, um dois e três da camada de entrada estão recebendo do meio externo o valor 0.

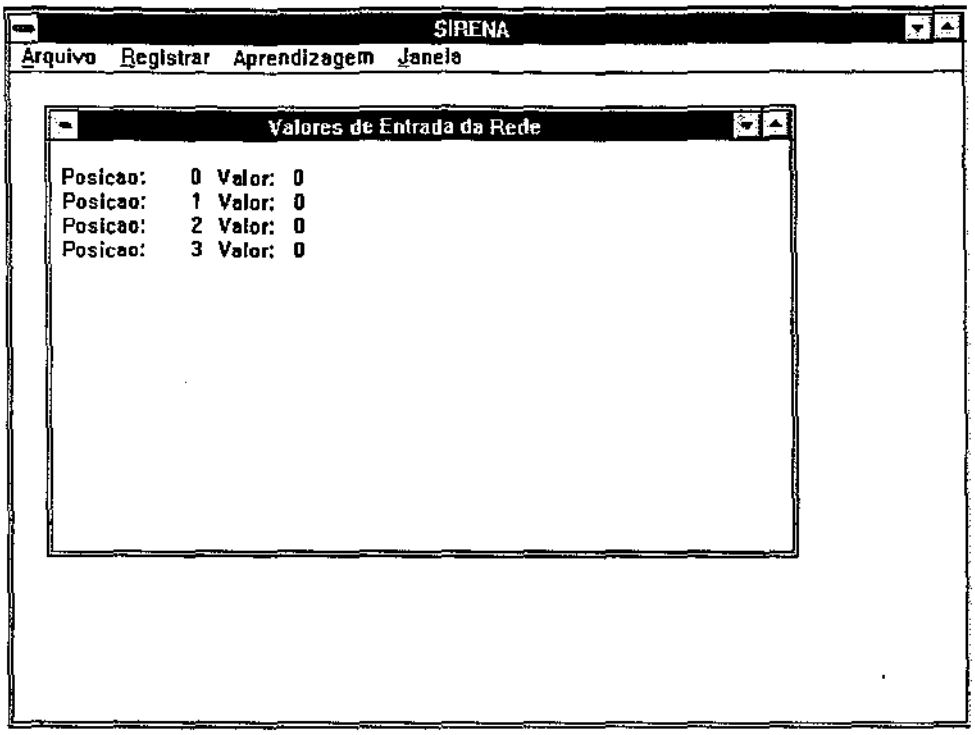
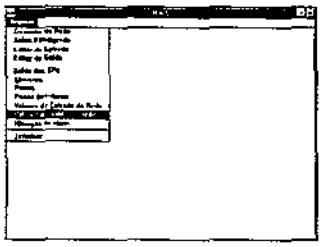


Figura V.10 - Janela Valores de Entrada da Rede.



V.3.6 - Janela Valores de Saída da Rede.

Análoga à anterior esta janela mostra os valores do vetor de saída da rede, isto é, o vetor que em cada uma das suas entradas contém o valor de saída de um EP da camada de saída da rede.

Como na RNA Exemplo a camada de saída apresenta somente um EP, a figura V.11 mostra somente o elemento de vetor de saída correspondente a este EP.

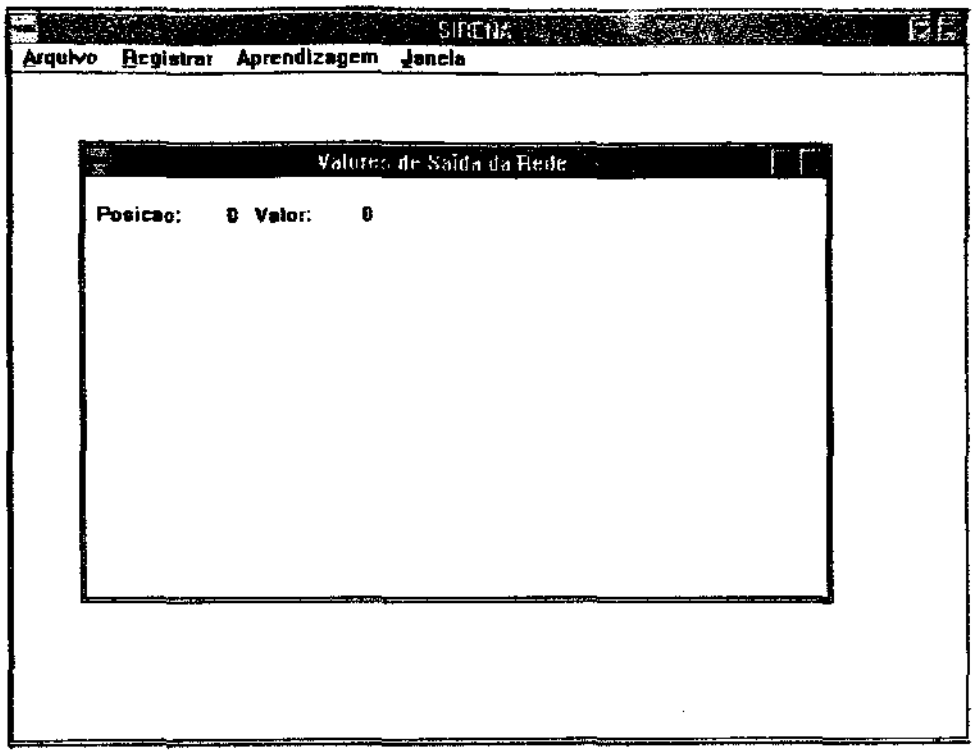
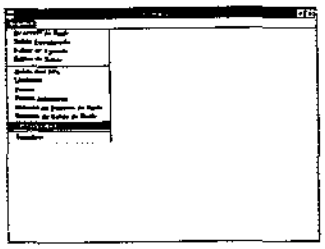


Figura V.11 - Janela Valores de Saída da Rede.



V.3.7 - Janela Situação da Rede.

Esta janela é a última das janelas textuais e objetiva mostrar qual é a distribuição atual dos EPs na RNA criada.

Ela apresenta a quantidade total de EPs na rede, a quantidade de EPs na camada de entrada, a distribuição dos EPs pelas camadas ocultas e a quantidade de EPs na camada de saída.

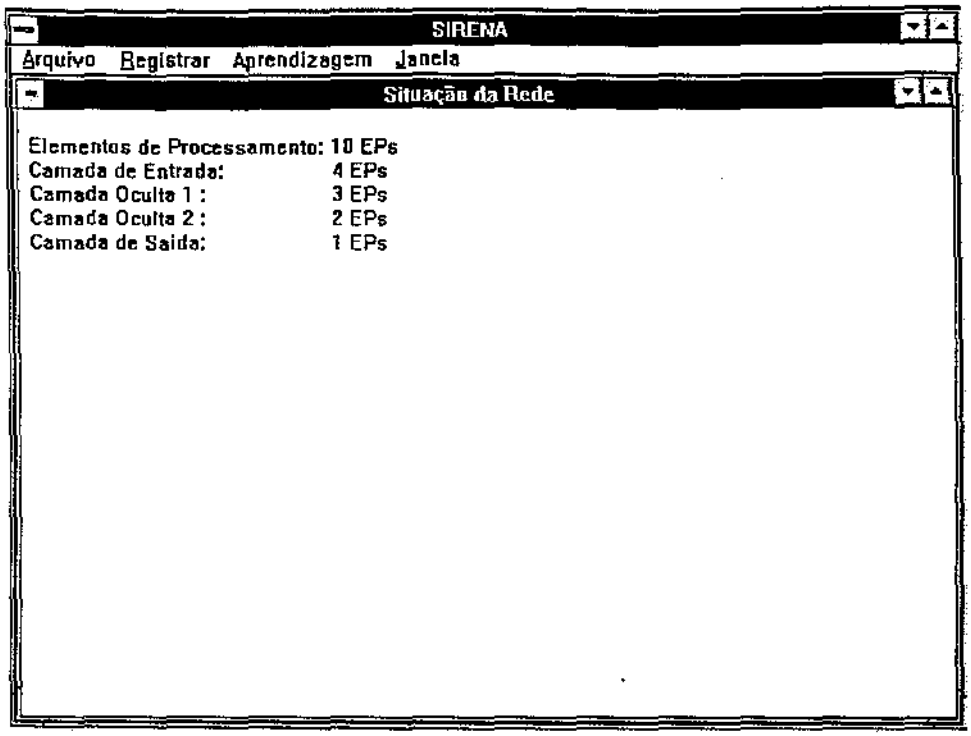


Figura V.12 - Janela Situação da Rede.

Descreveremos a seguir as janelas pertencentes ao segundo grupo de janelas do item Arquivo do *"menu"* principal: as janelas gráficas.

V.4 - JANELAS GRÁFICAS.

A este grupo de janelas, pertencem as janelas: Desenho da Rede, Editor de Entrada, Editor de Saída e Saída Estruturada.

Estas janelas caracterizam-se por apresentarem informações gráficas, como figuras geométricas e cores, além das informações textuais.

As janelas gráficas Editor de Entrada, Editor de Saída e Desenho da Rede são as únicas janelas que possibilitam interação extra com usuário além da permitida com a utilização do *"menu"* principal e da Caixa de Diálogo Principal que controla as iterações do simulador.

As janelas gráficas do Sirena podem ser vistas na figura V.13.

Por não possuírem barras de rolagem, os conteúdos das janelas gráficas Editor de Entrada, Editor de Saída e Saída Estruturada possuem forma variável, isto é, o conteúdo dessas janelas aumenta ou diminui conforme o tamanho da janela aumenta ou diminui.

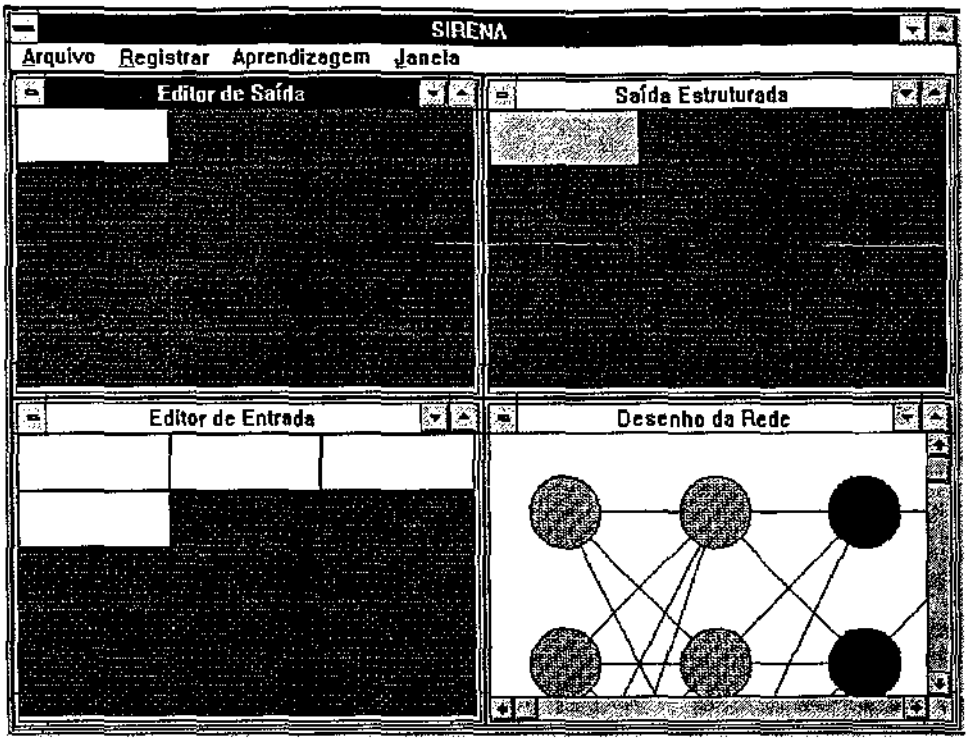


Figura V.13 - Janelas gráficas.

Tal característica permite que essas janelas possam ser ajustadas de forma a favorecer a visualização completa das mesmas quando várias janelas da ferramenta estiverem simultaneamente ativas.

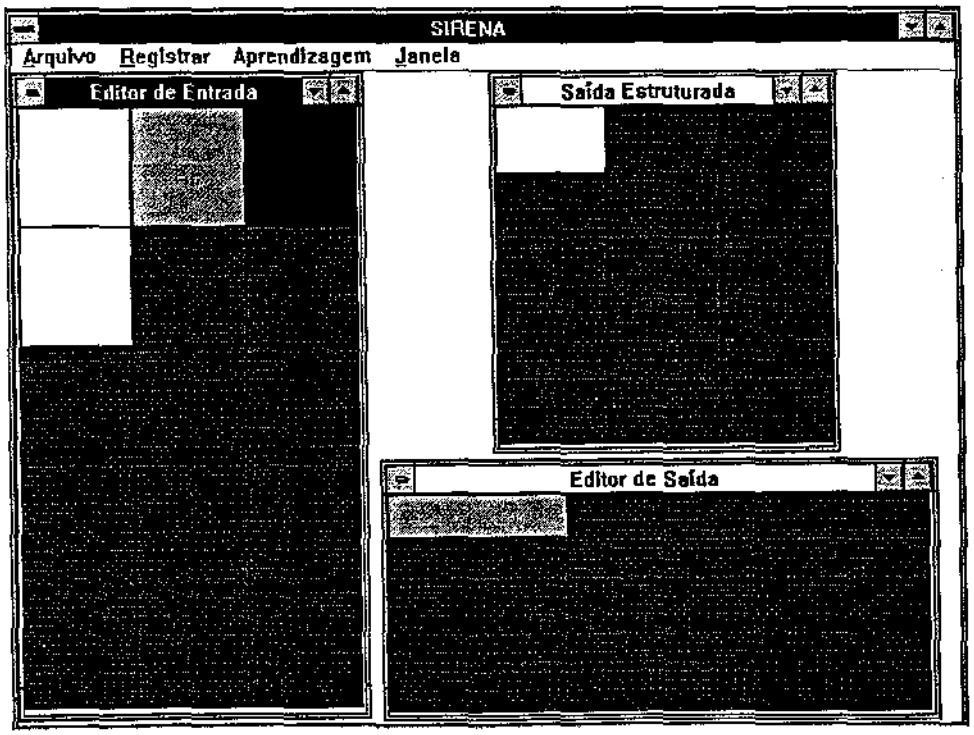
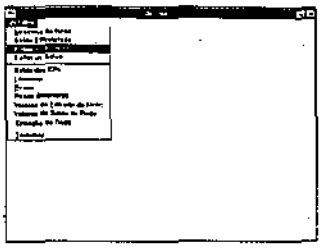


Figura V.14 - Forma variável das janelas gráficas.

A seguir descreveremos com mais detalhes cada uma das quatro janelas gráficas e suas características.



V.4.1 - A Janela Editor de Entrada.

A janela Editor de Entrada é a janela que permite ao usuário entrar com os valores desejados para o vetor de entrada da rede. Optou-se por fazer uma janela onde o usuário não tenha que entrar com os valores numericamente, mas sim, através de cores que representam valores específicos.

Esta janela é utilizada tanto para o processo de aprendizagem quando são fornecidos os exemplos a serem "treinados", quanto para o processo de inferência na introdução dos padrões de entrada da rede.

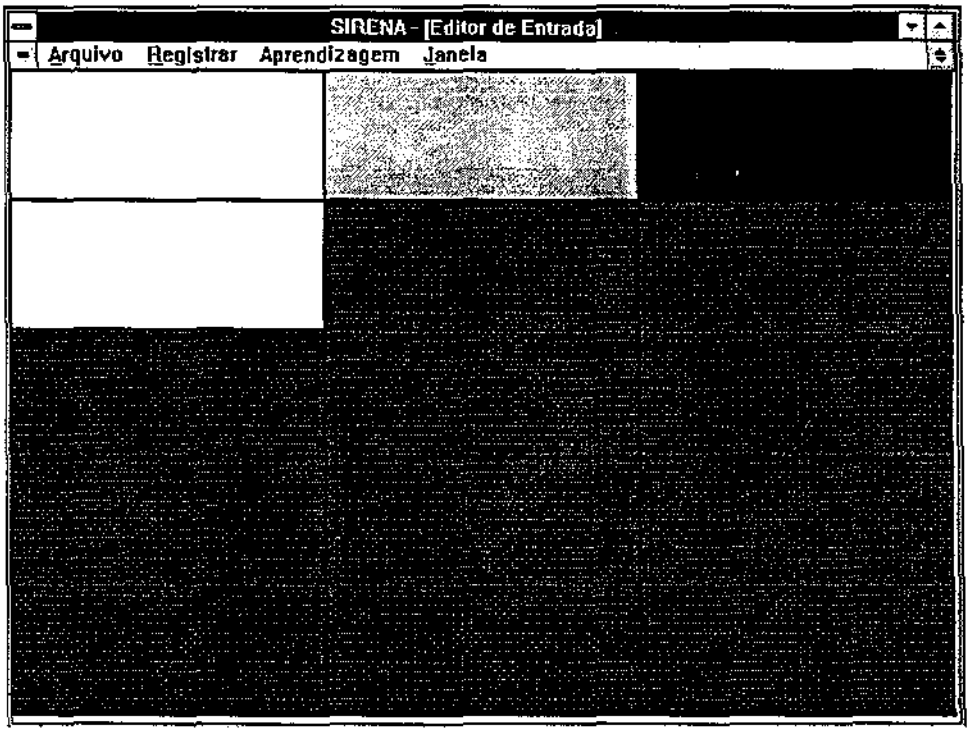


Figura V.15 - Janela Editor de Entrada.

Os valores do vetor de entrada da rede podem ser observados tanto pela janela textual Valores de Entrada da Rede como pela janela gráfica Editor de Entrada, mas só podem ser modificados através da última.

Apesar de grande número das aplicações de RNAs utilizar apenas dois valores (0 e 1) para representar as informações do mundo real, decidiu-se permitir a inclusão de um terceiro estado que ficasse equidistante dos outros dois e que, portanto, representasse uma situação intermediária. A esse terceiro estado atribuiu-se o valor 0,5 .

A janela Editor de Entrada apresenta retângulos que representam os EPs da camada de entrada da rede, são em número de quinze que é o tamanho máximo de EPs por camada na rede permitido pelo simulador na atual implementação.

A cada EP criado, um retângulo da janela, muda da cor cinza escuro, que representa a não existência de EP, para uma outras três cores que representam o valor de entrada desejado para o EP.

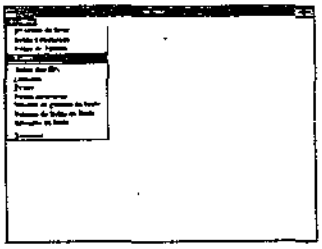
São três as cores implementadas para os valores de entrada desejados: branca, cinza e preta. A estas cores foram associados valores escolhidos de forma a permitir ao usuário representar qualitativamente a informação de entrada da rede em três categorias, a de valores baixos (cor branca com valor 0), a de valores médios (cor cinza com valor 0,5) e a de valores altos (cor preta com valor igual a 1).

Os EPs da camada de entrada estão associados aos retângulos seguindo-se o sentido de cima para baixo e da esquerda para direita, isto é, o EP na posição zero será representado pelo retângulo superior esquerdo e o EP na posição quatorze será representado pelo retângulo inferior direito.

A figura V.15 mostra a camada de entrada da RNA Exemplo formada por quatro EPs sendo que os valores de entrada desejados são respectivamente 0, (0,5), 1 e 0.

Apertando-se o *"mouse"* em um retângulo cinza escuro, um "beep" será soado indicando a tentativa de ajuste de valor em um EP inexistente. Apertando-se em um retângulo branco ele se tornará cinza claro, apertando-se em um cinza claro ele se tornará preto e finalmente apertando-se em um preto ele se tornará branco.

O ajuste dos valores de entrada em um EP, portanto, se dá de forma cíclica através de ajustes sucessivos em um mesmo retângulo.



V.4.2 - Janela Editor de Saída.

Esta janela permite o ajuste dos valores de saída desejados para a rede, valores estes pertencentes ao vetor de saída da rede e que também são mostrados pela janela textual Valores de Saída da Rede.

A janela Editor de Saída é semelhante à janela Editor de Entrada tanto na forma quanto na filosofia, apresentando, portanto, também quinze retângulos, um para cada EP da camada de saída,

com as três cores representando os valores de saída desejados para cada EP (a cor branca representando o valor de saída 0, a cinza claro o valor 0,5 e a preta o valor 1).

A forma de ajuste do valor de saída desejado também se dá de maneira cíclica apertando-se o botão do *"mouse"* quando o cursor estiver localizado sobre o retângulo.

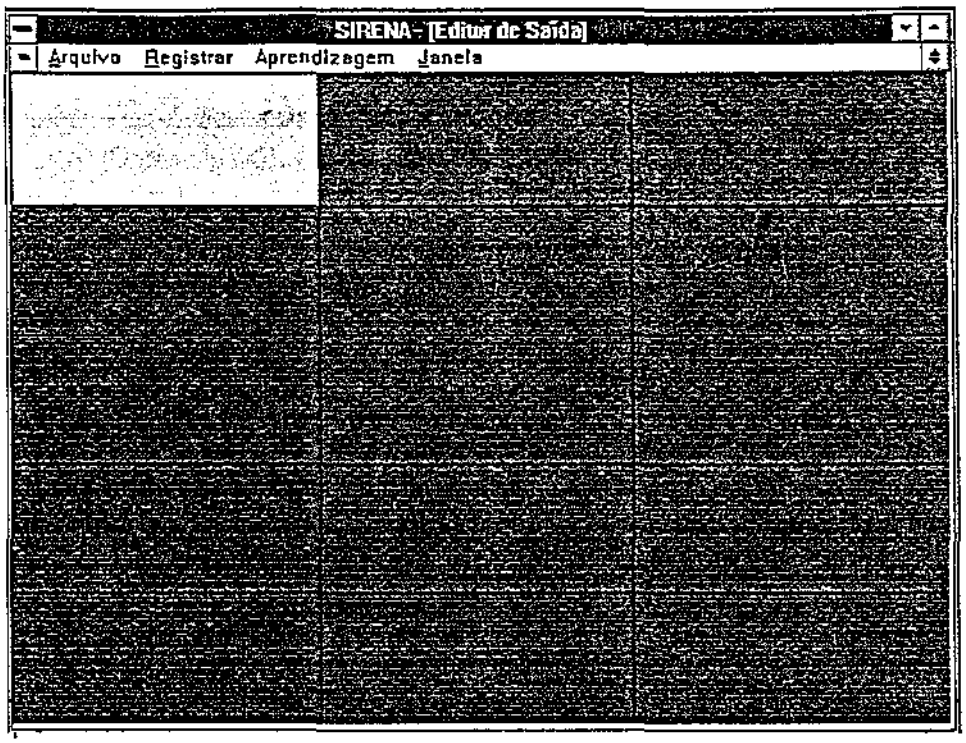
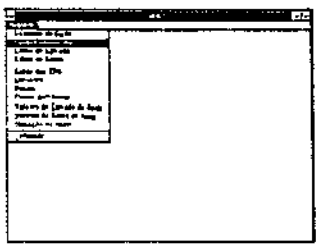


Figura V.16 - Janela Editor de Saída.

A figura V.16 mostra o EP da camada de saída da RNA Exemplo como valor de saída desejado respectivamente igual a (0,5), representado pela cor cinza.



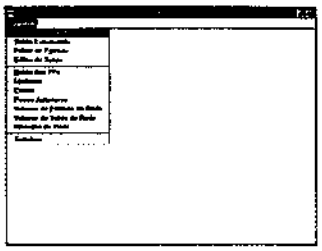
V.4.3 - Janela Saída Estruturada.

A Janela Saída Estruturada apresenta através de cores o valor de saída da função de transferência dos EPs da camada de saída.

As cores são as mesmas das duas janelas anteriores, mas seus valores são diferentes das mesmas pois para um dado valor de saída x, a cor do retângulo correspondente será branca se $0 <= x <= 0.1$, cinza claro se $0.1 < x < 0.9$ e preta se $0.9 <= x <= 1$.

Mesmo tendo a mesma aparência das janelas Editor de Entrada e Editor de Saída, a janela Saída Estruturada não permite ao usuário a modificação do seu conteúdo, pois seu objetivo é apresentar os valores de saída através de cores e mostrar se os mesmos estão formando os padrões geométricos desejados ou não.

Enquanto que a janela Editor de Saída mostra o padrão de saída desejado, a janela Saída Estruturada mostra o padrão efetivamente obtido pela rede através de sua camada de saída.



V.4.4 - Janela Desenho da Rede.

A janela Desenho da Rede é, dentre todas as janelas da ferramenta, aquela que possibilita maior interação com usuário.

Aparentemente sua única atividade é a de mostrar o grafo que representa a RNA simulada, isto é, a visualização da rede, mas na verdade a janela realiza outras tarefas como:

- Ajustar o tamanho dos EPs no grafo.
- Ajustar a forma geométrica dos EPs no grafo.
- Permitir alterações de tamanho dos EPs no grafo.
- Permitir alterações de cor dos EPs no grafo.
- Seleção de EPs no grafo.
- Criação e eliminação de EPs.
- Modificação dos valores de saída e limiar de cada EP.
- Criação e eliminação de ligações entre os EPs.
- Modificação dos valores do peso e do peso anterior de cada ligação da rede.

A janela Desenho da Rede realiza em termos gerais, as tarefas de visualização, construção e modificação da RNA, tornando-se, portanto, a principal janela da ferramenta.

Descreveremos agora, em maiores detalhes, as tarefas acima descritas.

V.4.4.1 - Visualização da rede.

A Janela Desenho da Rede mostra a RNA construída no módulo Simulador através de um grafo que segue sempre o seguinte formato:

- As camadas da rede estão dispostas verticalmente, sendo que a camada de entrada é sempre a mais esquerda e a de saída a mais a direita.
- As ligações embora orientadas, não são representadas como tal, ficando subentendido o sentido das mesmas da esquerda para direita.
- O primeiro EP da camada de entrada, fica sempre localizado na porção superior esquerda da janela.
- Todos os EPs que estão na mesma posição dentro da camada, mas em camadas diferentes, localizam-se na mesma reta horizontal.
- A distância entre as camadas e a distância entre EPs de uma mesma camada são constantes, mas não necessariamente iguais.

Para a RNA Exemplo a Janela Desenho da Rede mostra seu grafo como na figura V.17.

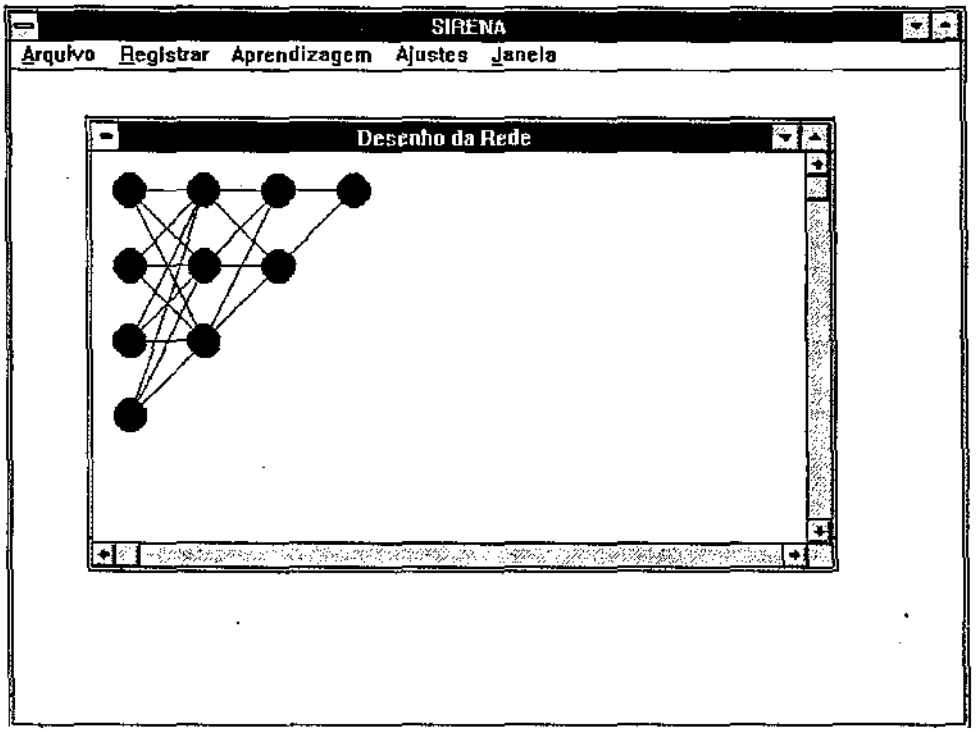


Figura V.17 - Padrão de desenho do grafo.

Como o tamanho dos EPs no grafo pode ser alterado, o grafo da rede pode não ser visto em toda sua extensão dentro da janela, fazendo-se necessário neste caso, o uso das barras de rolagem como na figura V.18.

Com elas o usuário poderá visualizar qualquer parte do grafo que representa a rede não importando o seu tamanho desde que o simulador o suporte.

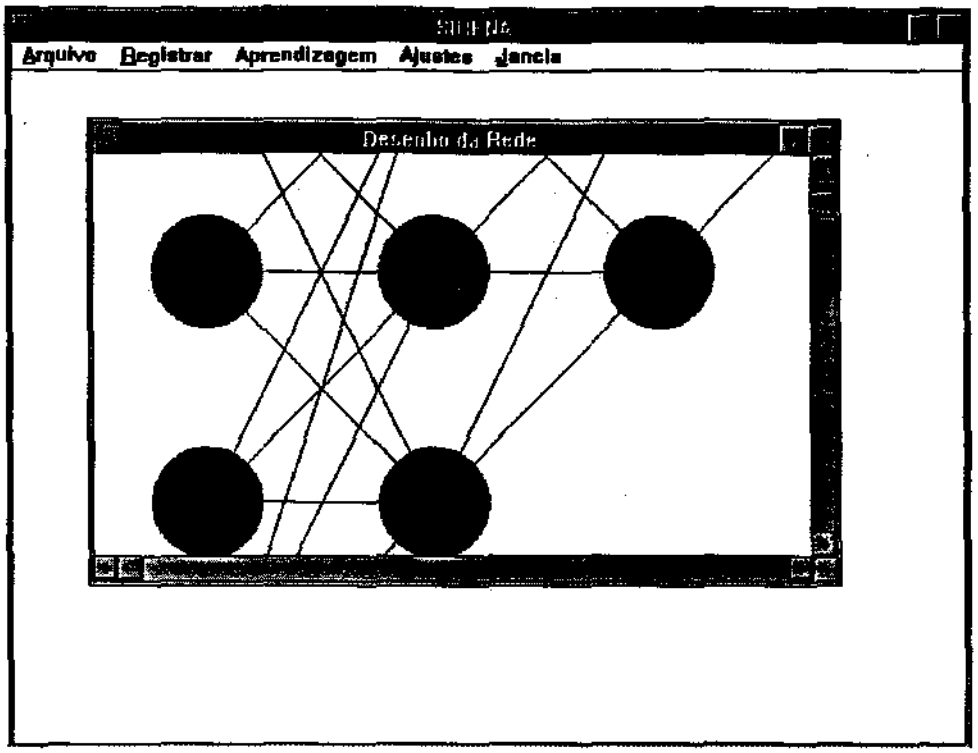


Figura V.18 - Utilização das barras de rolagem.

V.4.4.2 - Ajuste do Tamanho dos EPs no Grafo.

A Janela Desenho do Grafo é única janela que quando ativada, provoca modificação no número de itens do *"menu"* principal, que passa de quatro para cinco itens como mostrado na figura V.19.

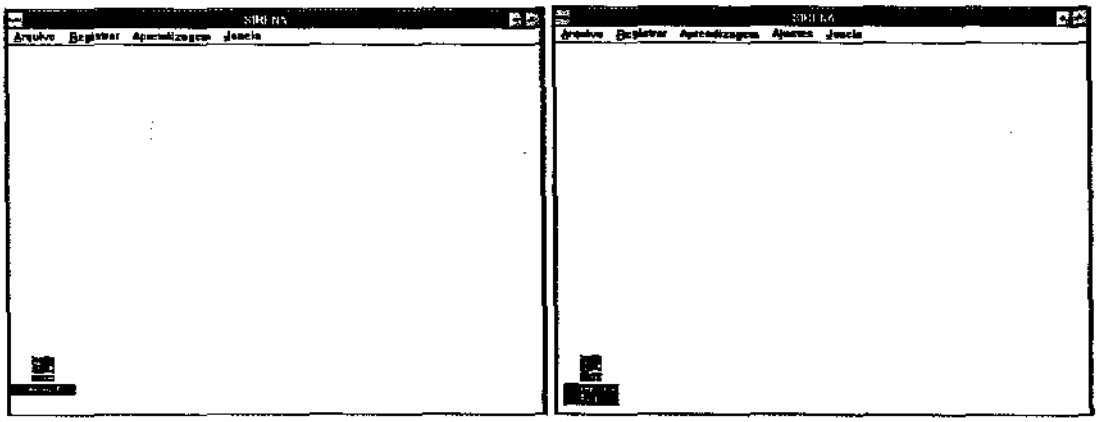


Figura V.19 - Comparação dos dois tipos de *"menu"*.

O item adicional de nome "Ajustes" tem como função a criação da caixa de diálogo de mesmo nome, veja figura V.20, que dentre outras coisas permite a escolha do tamanho dos EPs a serem desenhados no grafo.

O objetivo da variação de tamanho dos EPs é de permitir a visualização em detalhes de um ou mais EPs, ou da rede inteira sem o uso das barras de rolagem.

Como o tamanho do grafo não varia conforme o tamanho da janela, quando quisermos observar uns poucos EPs, escolheremos uma ampliação grande; quando o tamanho da janela é grande e quisermos ver o grafo inteiro, escolheremos uma ampliação média e quando o tamanho da janela é pequeno, devido à presença de outras janelas da ferramenta, e ainda assim quisermos ver a rede inteira, escolheremos então uma ampliação pequena.

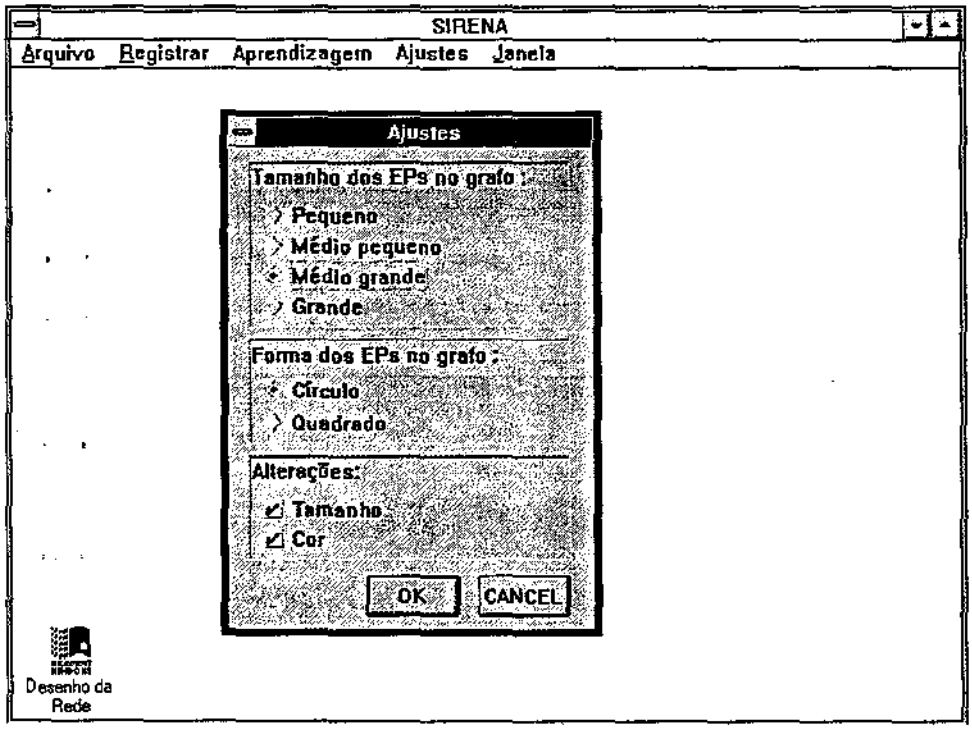


Figura V.20 - Caixa de Diálogo Ajustes.

O grupo da caixa de diálogo "Tamanho dos EPs no grafo", é o responsável pela opção de escolha da ampliação desejada, permitindo quatro opções: grande ampliação, média grande ampliação, média pequena ampliação e pequena ampliação.

O valor médio grande é o valor padrão da caixa de diálogo, isto é, é o valor com que o grafo é desenhado quando a janela Desenho do Grafo é apresentada.

As diferenças relativas dos tamanhos dos EPs no grafo podem ser vistas na figura V.21.

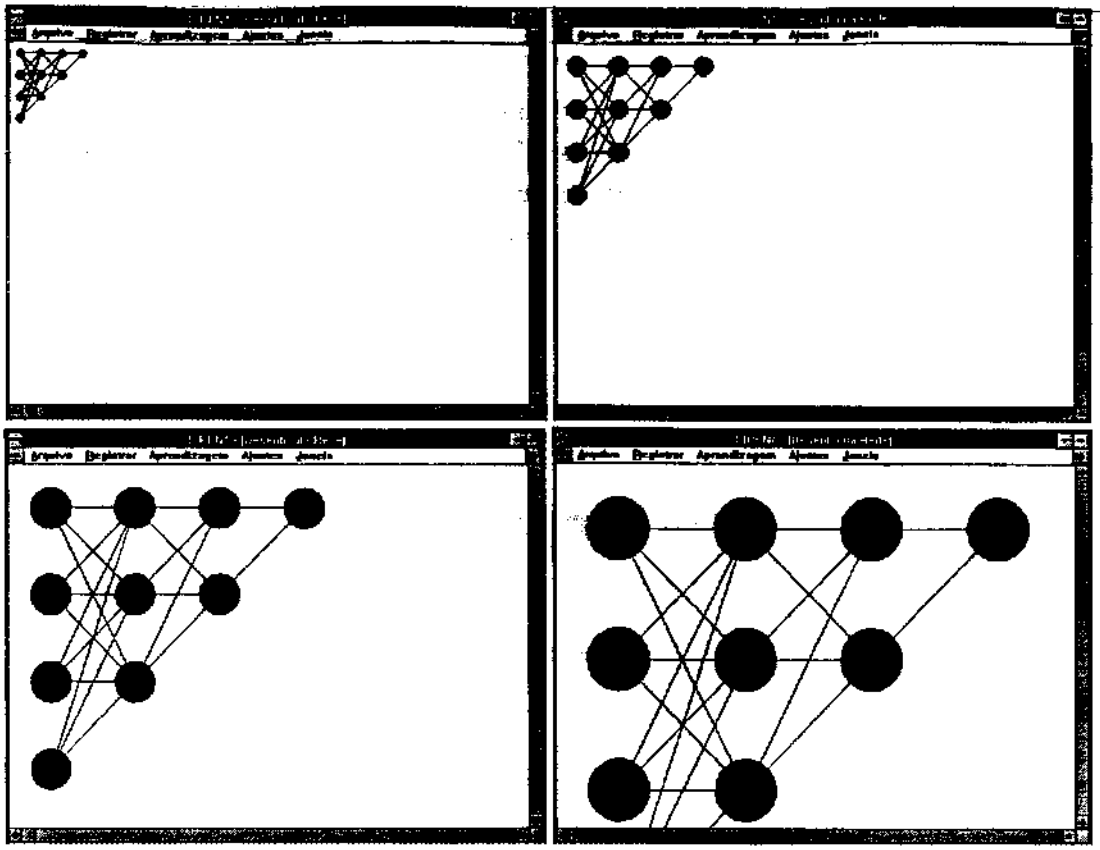


Figura V.21- Comparação das ampliações da rede.

V.4.4.3 - Ajuste da Forma Geométrica dos EPs no Grafo.

Esta é mais uma opção da Caixa de Diálogo Ajustar, que em seu grupo "Forma dos EPs no grafo", permite ao usuário escolher que os EPs da rede sejam desenhanhos no grafo como uma circunferência ou como um quadrado, tendo como único objetivo desta escolha, a comodidade visual do usuário.

A diferença de apresentação da rede pode ser vista na figura V.22.

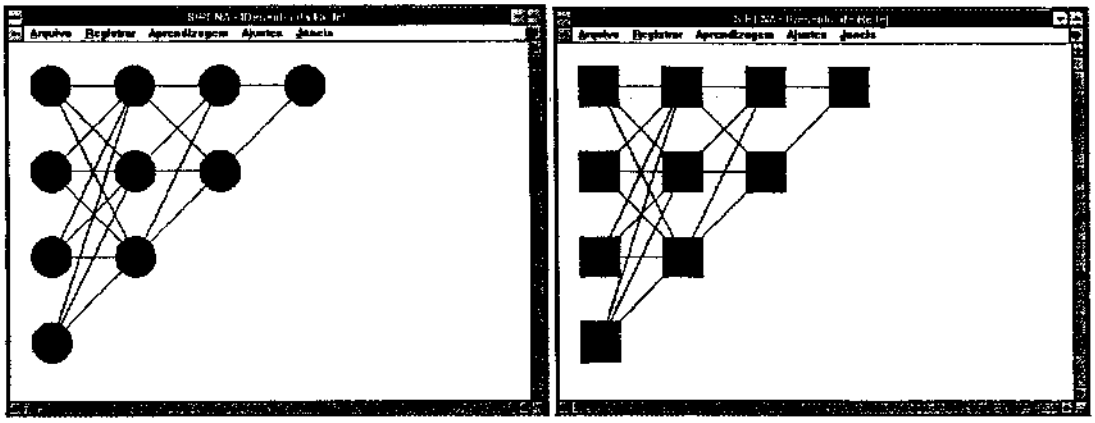


Figura V.22 - Formas geométricas da representação dos EPs.

V.4.4.4 - Alteração dos Tamanhos dos EPs no Grafo.

Até o momento falou-se que os EPs poderiam variar de tamanho, mas como uma maneira de ampliação da visualização da rede, isto é, os EPs ficariam maiores ou menores conforme a conveniência do usuário, mas todos eles teriam o o mesmo tamanho no grafo.

Podemos através da opção Tamanho do grupo "Alterações", como mostrado na figura V.23, permitir que cada EP da rede varie de tamanho conforme o seu valor de saída, crescendo de tamanho conforme o crescimento deste valor.

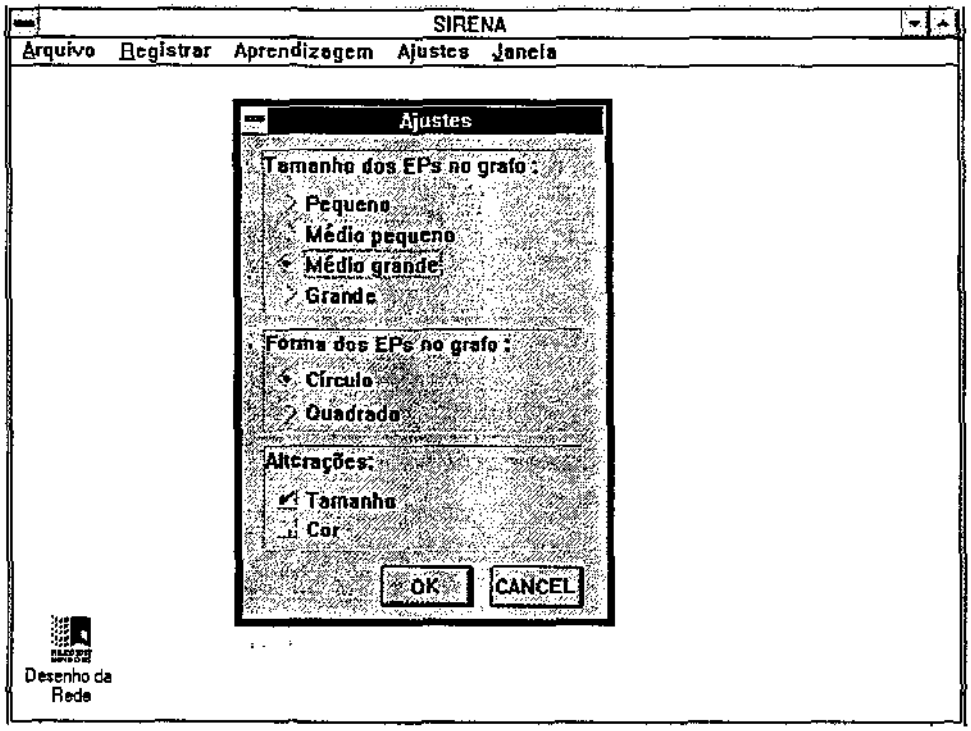


Figura V.23 - Ajuste para alterações de tamanho.

Com esse mecanismo, podemos verificar a variação de comportamento de cada EP em relação a si mesmo e em relação aos outros EPs da rede.

A alteração dos tamanhos dos EPs no grafo ainda está sujeita ao processo de ampliação do tamanho da rede na janela, isto é, mesmo tendo tamanhos diferentes entre si, os EPs podem todos juntos, aumentarem ou diminuírem seus tamanhos em uma certa quantidade a fim de facilitar a visualização dos mesmos.

Na figura V.24 mostramos a RNA Exemplo em que os EPs variam seus valores de saída de 0,1 a 1.

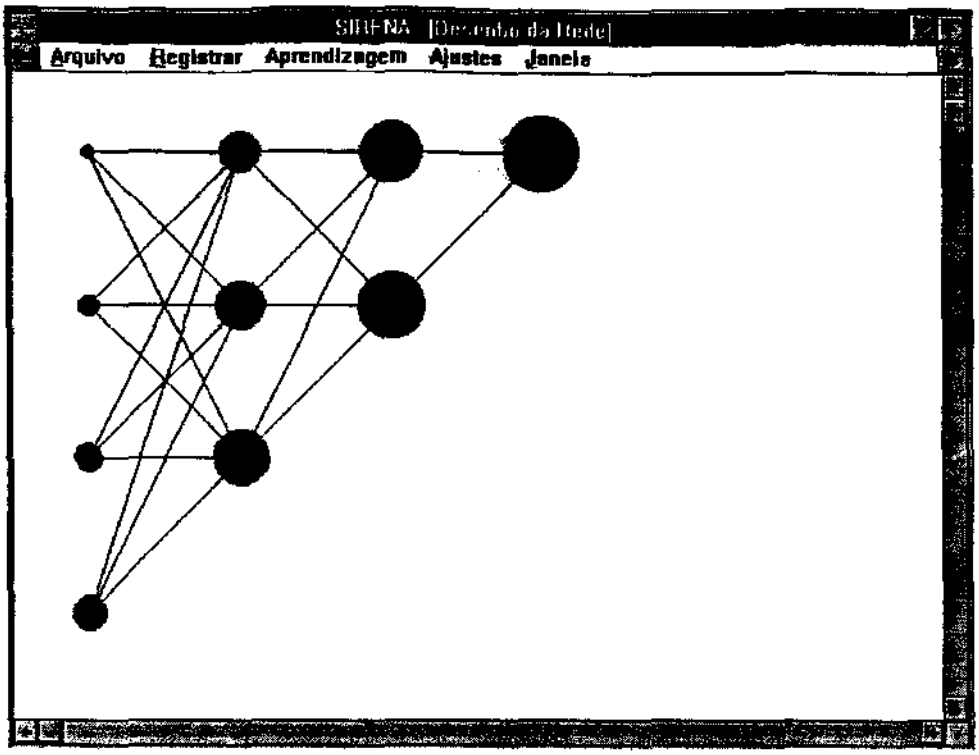


Figura V.24 - Rede com alterações de tamanho.

V.4.4.5 - Alterações de Cor dos EPs no Grafo.

De maneira semelhante à variação do tamanho dos EPs no grafo, podemos permitir através da opção Cor do grupo "Alterações" (figura V.23), que cada EP seja preenchido com uma cor diferente no grafo da rede conforme seu valor de saída.

Não é pretensão que o usuário faça uma associação cor/valor de saída do EP, mas sim de que ele consiga relacionar a variação deste valor com seu valor anterior ou com o valor dos demais EPs.

Na figura V.25 vemos a variação de cores para a rede exemplo que tem os seus EPs com valores de saída variando de 0,1 a 1.

A ferramenta permite que haja variação de tamanho e de cor dos EPs simultaneamente originando um grafo como o da figura V.26.

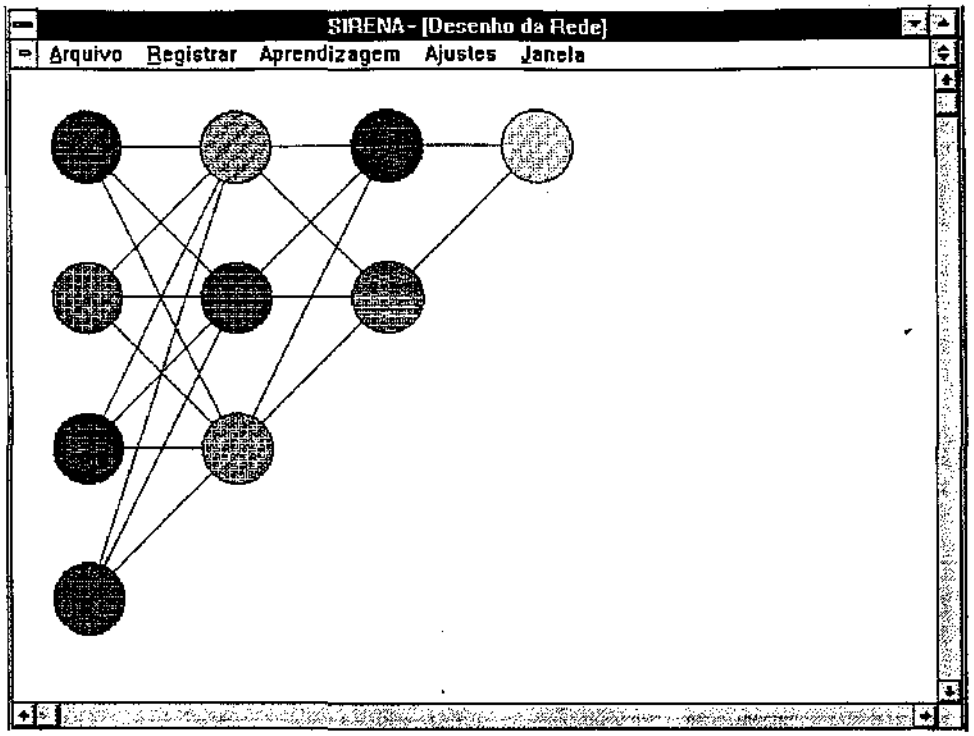


Figura V.25 - Alterações de cor sem alterações de tamanho.

Este é o ajuste que mais favorece a distinção dos valores de saída dos EPs por parte do usuário, sendo este, portanto, o ajuste padrão.

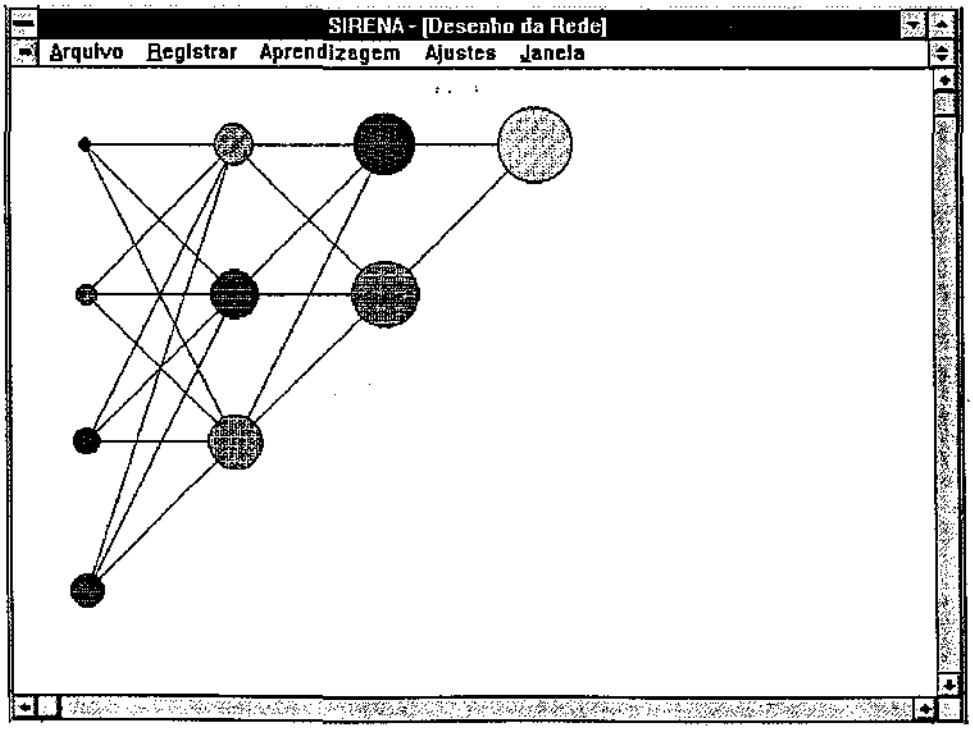


Figura V.26 - Alterações de cor e de tamanho.

V.4.4.6 - Seleção dos EPs no Grafo.

Toda a manipulação dos EPs e de suas ligações no Sirena, só pode ser feita após a seleção de pelo menos um EP na janela Desenho da Rede.

A seleção de um EP se dá pelo pressionamento esquerdo do *"mouse"* em uma região próxima ao EP desenhado na janela. Chamamos tal região de Retângulo de Seleção.

Este retângulo aparece de forma destacada no grafo quando um EP é selecionado como mostrado na figura V.27.

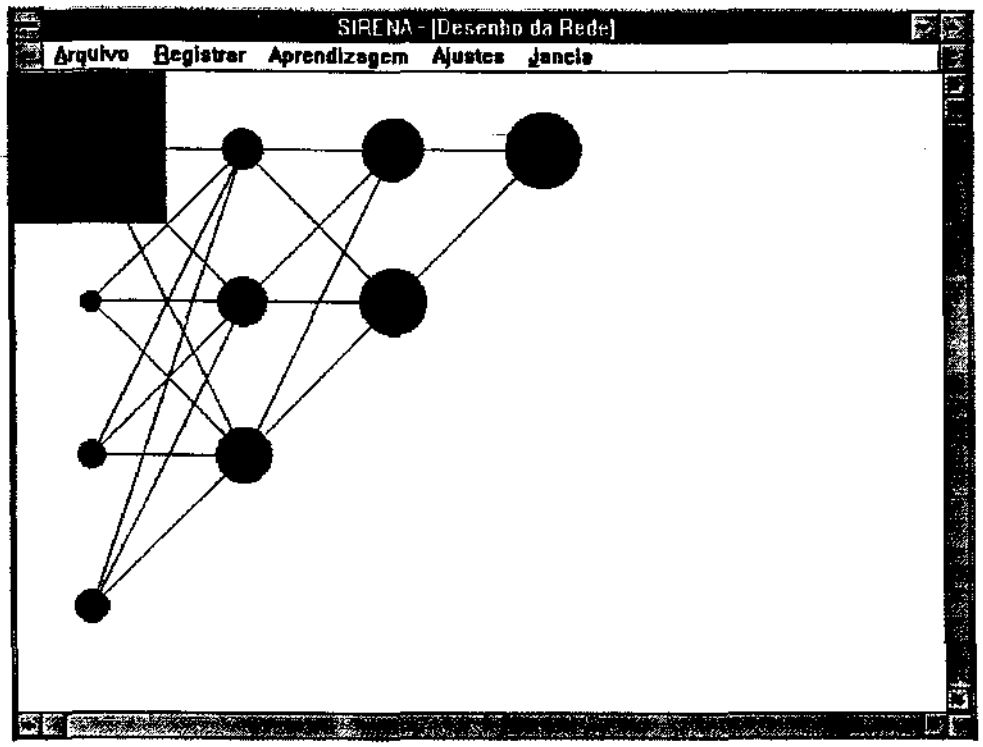


Figura V.27 - Seleção de um EP.

Sempre que um EP está selecionado e desejamos cancelar a opção, basta apertarmos o *"mouse"* novamente em seu Retângulo de Seleção que se encontra ativo, para que o mesmo se apague, indicando que o EP não está mais selecionado.

Quando dois EPs estão ligados ou ainda, quando estão em posições que permitam a criação de uma ligação entre eles, isto é, em camadas sucessivas da rede, então podemos selecionar os dois EPs ao mesmo tempo, como na figura V.28, para que possamos criar, eliminar ou modificar o valor desta ligação.

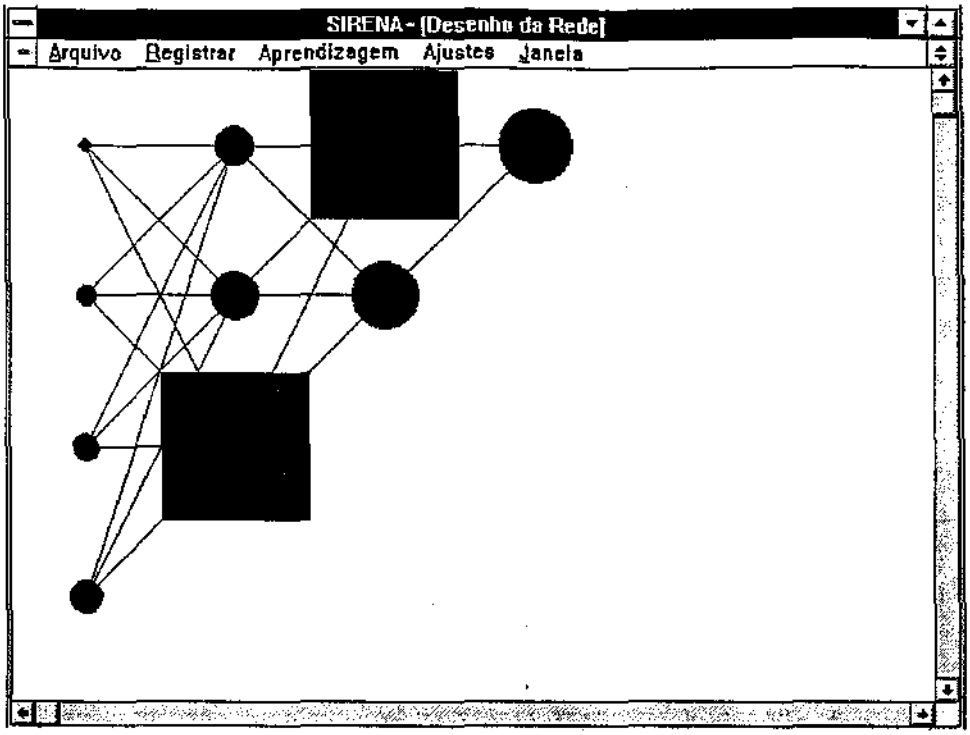


Figura V.28 - Seleção simultânea de dois EPs.

Quando um EP está selecionado e tentarmos selecionar outro que não está nem na camada anterior nem na posterior do EP selecionado, como o que ocorre no caso anterior, o primeiro EP deixará de ser selecionado e o segundo passará a ser o EP selecionado (veja figura V.29).

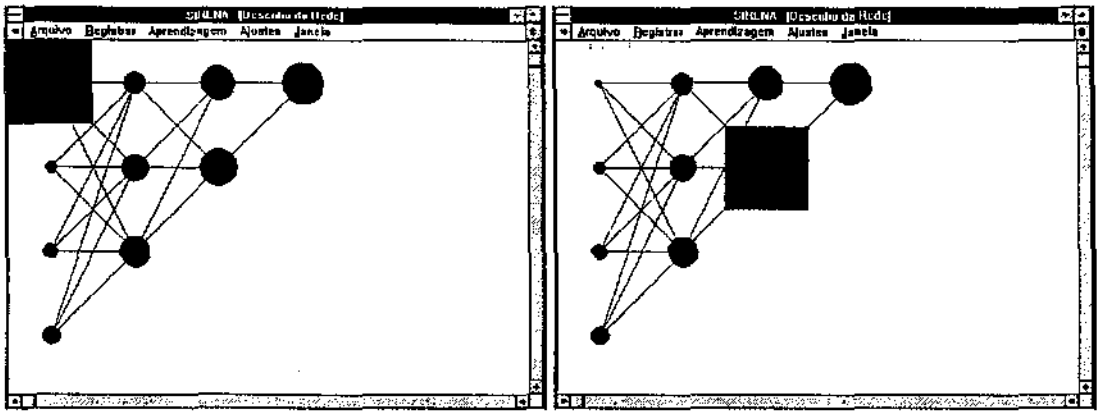


Figura V.29 - Alternância de seleção entre EPs.

No caso de termos dois retângulos simultaneamente selecionados, podemos cancelar a seleção ou de um dos EPs, clicando-se em seu Retângulo de Seleção, ou dos dois EPs sucessivamente, cancelando a o seleção de cada um.

Podemos ainda cancelar a seleção de dois EPs simultaneamente caso selecionemos um terceiro EP como mostra a figura V.30 .

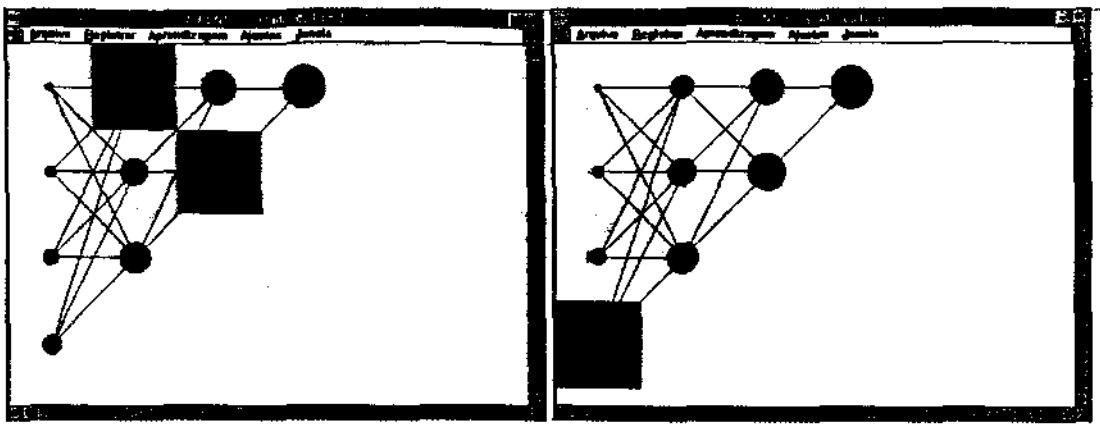


Figura V.30 - Alternância de seleção entre EPs.

A ferramenta nos permite selecionar a posição de um EP que ainda não foi criado apenas pressionando-se o botão do *"mouse"* em sua localização.

O retângulo selecionado, porém, poderá não corresponder ao desejado pois o desenho do grafo não permite EPs que não sigam os padrões especificados na seção "Visualização da Rede".

Por esse motivo se selecionarmos a posição de um EP não criado em uma camada da rede já existente, o retângulo selecionado será o do primeiro EP ainda não criado da camada, isto é, o EP ainda não criado cuja posição dentro da camada seja a menor possível. Ele estará, portanto, o mais alto possível dentro da janela naquela camada (veja figura V.31).

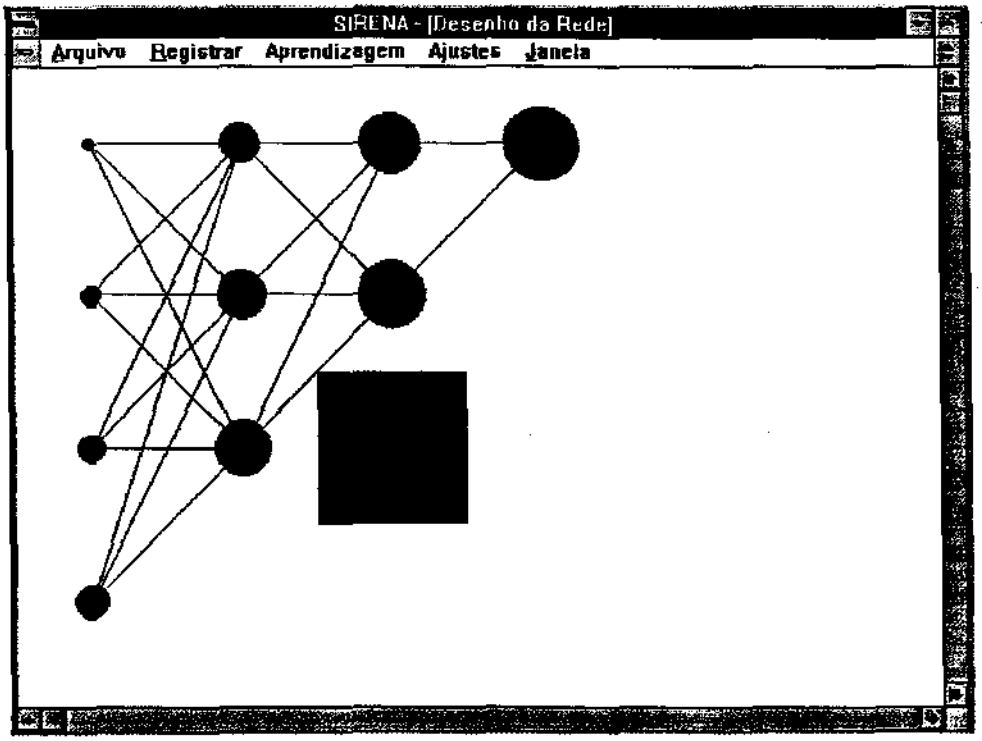


Figura V.31 - Seleção de posição para criação de EP na terceira camada.

Agora se quisermos selecionar um EP não criado em uma camada da rede ainda não existente, estaremos criando uma nova camada de saída, então o retângulo selecionado será o correspondente ao primeiro EP da camada seguinte a de saída como mostra a figura V.32.

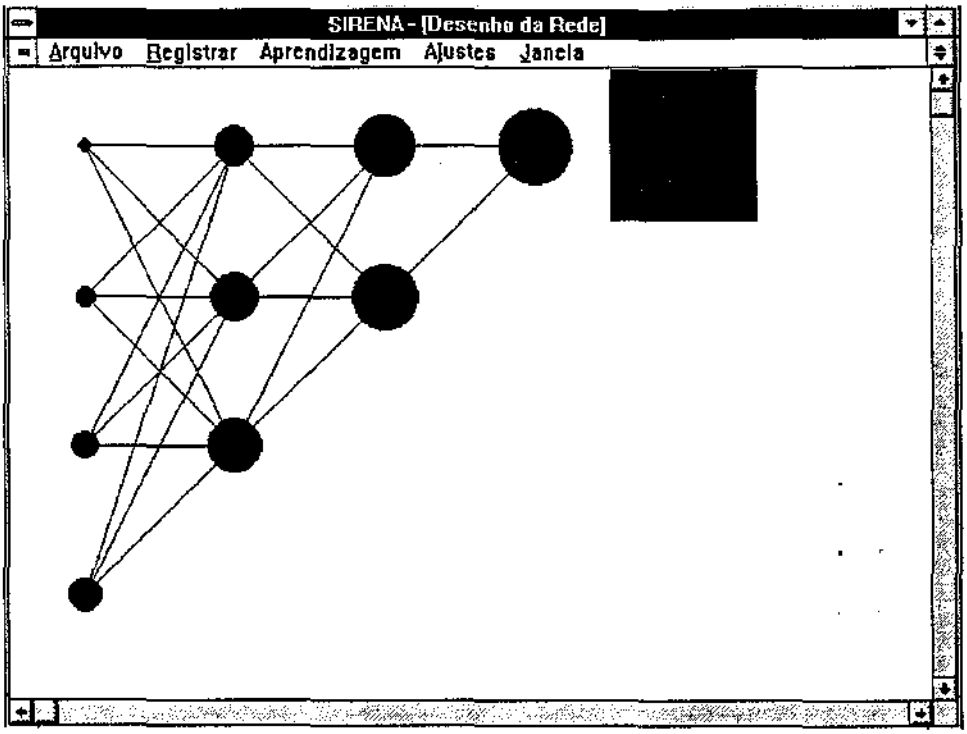


Figura V.32 - Seleção de posição para criação de nova camada.

V.4.4.7 - Alteração de um EP.

Para que possamos alterar o estado de um EP já criado, necessitamos primeiramente selecionar o EP desejado.

O segundo passo é pressionarmos em qualquer posição da janela o botão direito do *"mouse"* o que criará a **Caixa de Diálogo Cria e Altera EP.**

Esta caixa, que pode ser vista na figura V.33, apresenta três janelas de edição de texto denominadas Saída, Limiar e Delta que respectivamente permitem a alteração destes valores no EP selecionado. Os valores antigos destas grandezas do EP aparecem em suas respectivas posições.

Apertando-se o botão de OK, os valores novos serão os que estão mostrados nas janelas, estejam eles modificados ou não.

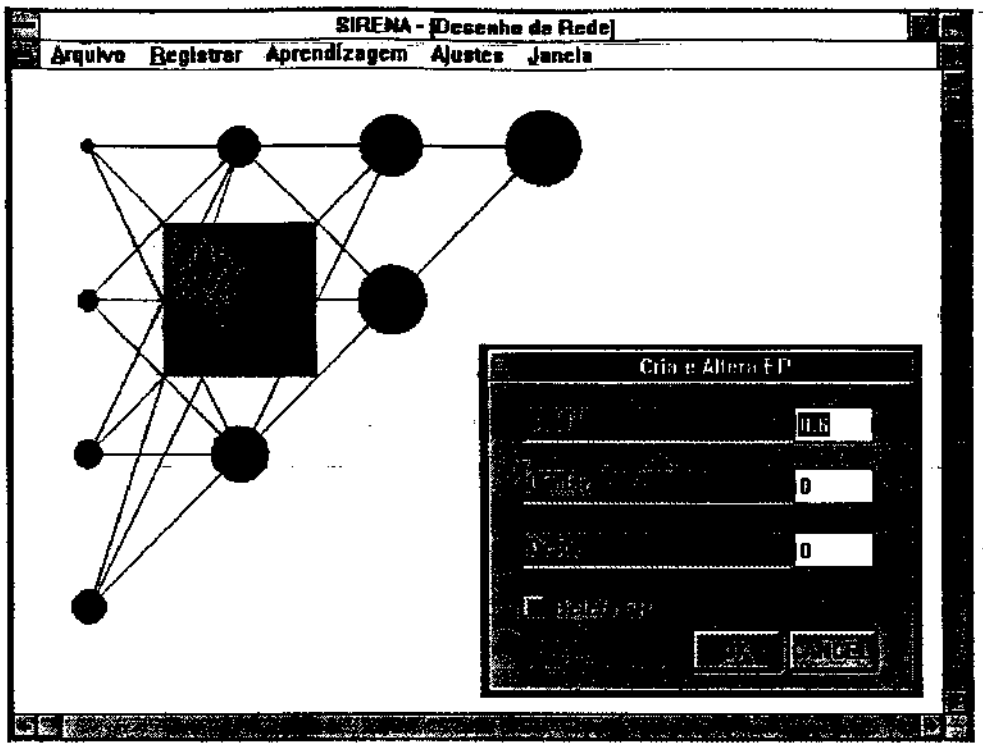


Figura V.33 - Caixa de Diálogo Cria e Altera EP.

V.4.4.8 - Criação de um EP.

A criação de um EP é feita, ativando-se o Retângulo de Seleção para a posição desejada do EP, ou a posição mais próxima da desejada conforme explicado na seção "Seleção dos EPs no Grafo".

Após termos ativado o retângulo, abrimos a Caixa de Diálogo Cria e Altera EP com o botão direito do *"mouse"* e preenchemos as janelas de edição de texto com os valores desejados para o EP a ser criado.

Essas janelas apresentam no momento de sua criação o valor 0, que é valor padrão para elas. Assim se só apertarmos o botão OK, o EP recém criado terá valor 0 de saída, 0 de limiar e 0 de delta.

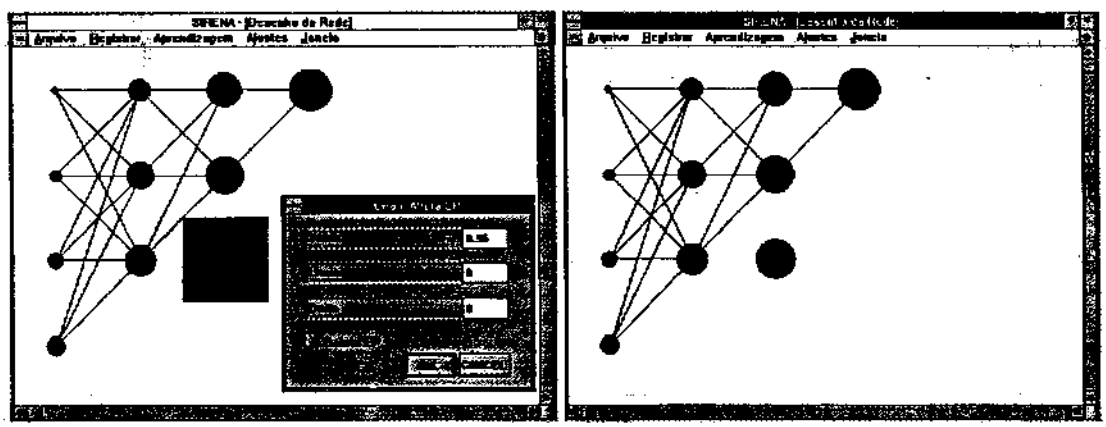


Figura V.34 - Criação de um novo EP.

V.4.4.9 - Alteração ou Eliminação de um EP.

A alteração dos valores de saída, limiar e delta associados a um EP da rede pode ser feita através da **Caixa de Diálogo Cria e Altera EP** após a seleção da mesma feita na janela Desenho da Rede.

A eliminação de um EP se dá de maneira semelhante, ou seja, através da seleção da mesma na janela Desenho da Rede e da opção "Deleta EP" na Caixa de Diálogo Cria e Altera EP. Após a confirmação da eliminação feita através do botão OK desta caixa de diálogo, a ferramenta:

- Elimina o EP selecionado e todas as ligações que chegam e saem do mesmo.
- Atualiza o número de camadas existentes, atribuindo, se necessário, a condição de camada de entrada ou de saída a uma outra camada.

Fica, porém, de responsabilidade do usuário eliminar EPs subseqüentes caso os mesmos só recebam informações do EP eliminado e o ajuste dos vetores de entrada e de saída caso as camadas de entrada e saída tenham algum de seus EPs eliminado.

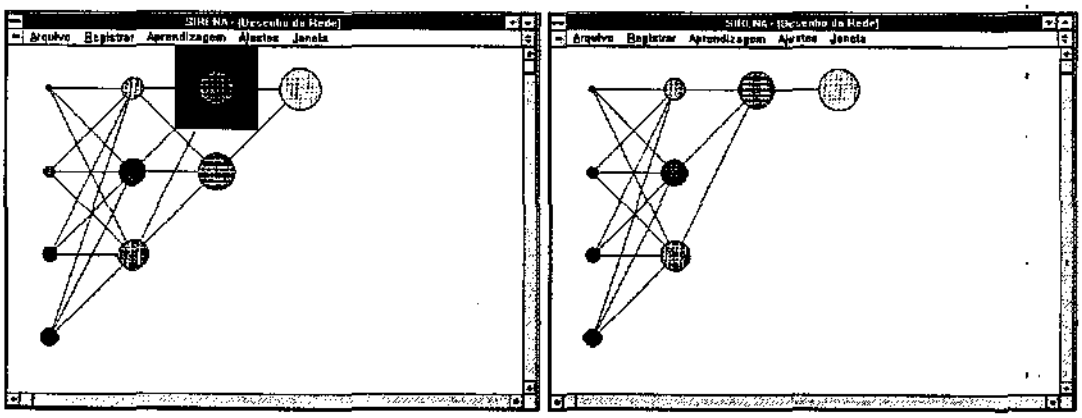


Figura V.35 - Eliminação de um EP.

V.4.4.10 - Criação de uma ligação.

Uma ligação pode ser criada quando selecionamos os dois EPs que determinarão a ligação e apertamos o botão direito do *"mouse"* criando a **Caixa de Diálogo Alterações no Peso**.

Essa caixa, que pode ser vista na figura V.36, apresenta duas janelas de edição de texto denominadas Peso e Último Peso que respectivamente nos permitem entrar com o valor atual do peso da ligação e com o valor anterior do mesmo.

Tal opção permite ao usuário, reproduzir fielmente um modelo de RNA no simulador no processo de construção da rede e também, caso a rede já esteja sendo simulada, acrescentar uma nova ligação que não destoe das demais por estar excessivamente facilitada ou inibida. O valor padrão de peso para uma ligação recém criada é de 0.

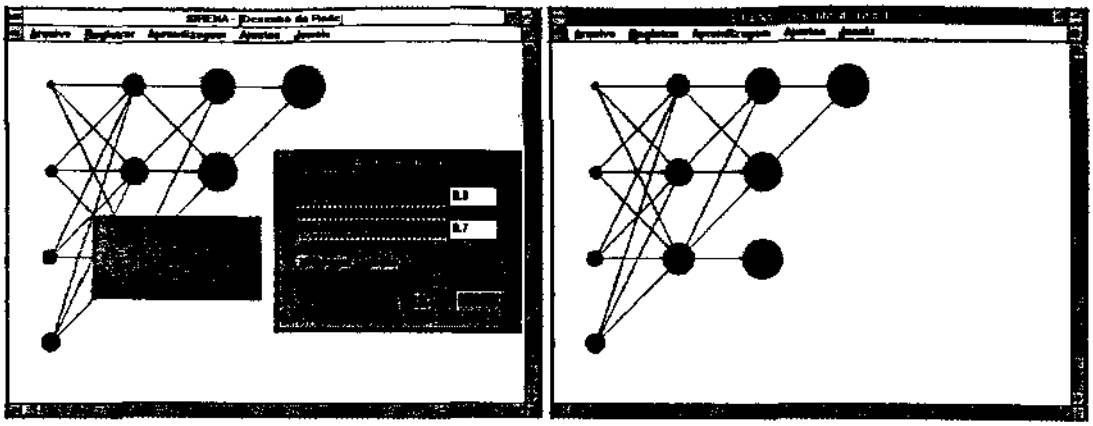


Figura V.36 - Criação de uma nova ligação.

V.4.4.11 - Alteração ou Eliminação de uma Ligação.

Para alterarmos ou eliminarmos uma ligação, devemos selecionar os EPs envolvidos e abrir a **Caixa de Diálogo Alterações no Peso**.

As alterações podem ser feitas, modificando-se os valores apresentados nas janelas Peso e Último Peso. A eliminação da ligação é feita selecionando-se a opção Apagar Ligação da Caixa de Diálogo Alterações no Peso

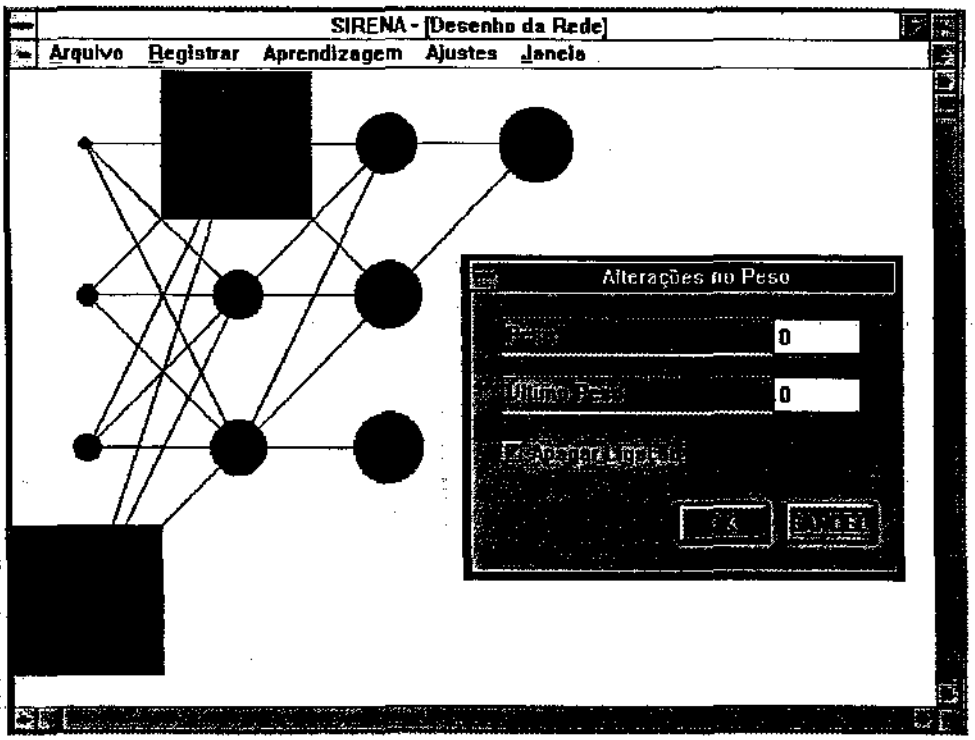


Figura V.37 - Caixa de Diálogo Alterações de Peso.

V.5 - Características Gerais.

A seguir mostraremos outras funcionalidades da ferramenta distribuídas em suas janelas e caixas e diálogo.

V.5.1 - O Armazenamento de Mais de Um ou Mais Exemplos de Entrada/Saída.

Quando uma RNA é criada na ferramenta, os valores dos vetores de entrada e de saída da rede são zerados, portanto, os exemplos de entrada e de saída desejados são o vetor zero de entrada e o vetor zero de saída, considerados os exemplos padrões, isto é, existem sempre mesmo que o usuário não se preocupe com eles.

Para o processo de inferência, a introdução de um exemplo de entrada pode ser feita simplesmente pela alteração da janela Editor de Entrada. Neste caso o exemplo padrão de entrada é temporariamente substituído pelo exemplo recém introduzido.

Para o processo de aprendizagem, porém, a simples alteração do exemplo padrão de entrada e do de saída não é suficiente, pois sem estar armazenado este novo padrão não é reconhecido pela ferramenta que só "enxerga" o exemplo padrão.

Tal fato acontece pois no módulo de aprendizagem do Sirena, cada iteração do algoritmo, recupera o próximo exemplo de entrada/saída armazenado nas estruturas do simulador, no caso o exemplo padrão.

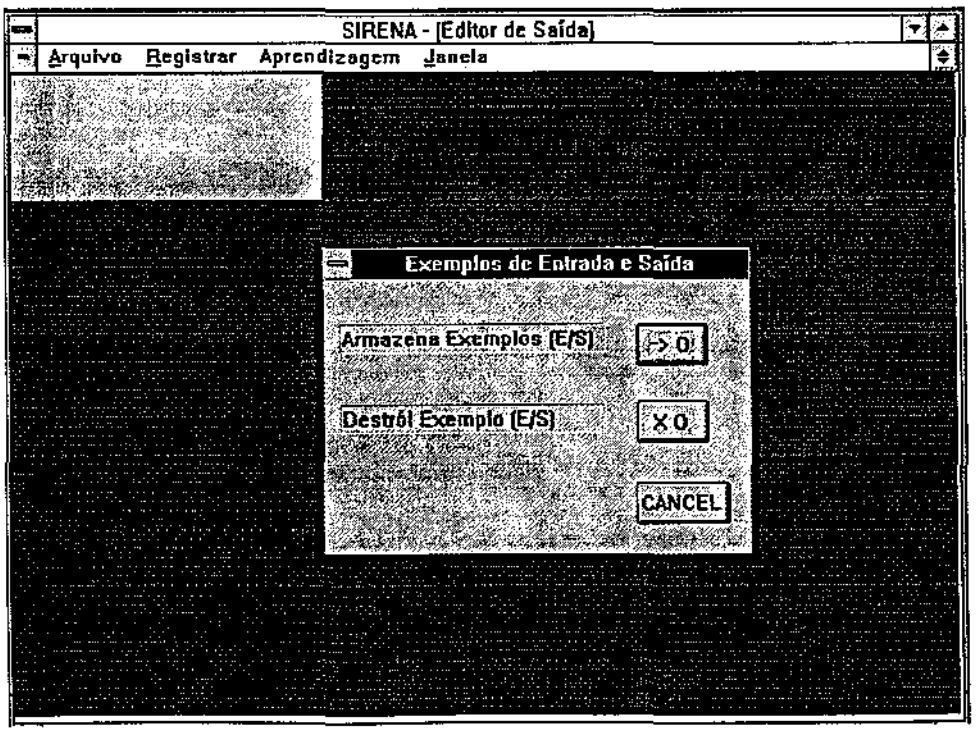


Figura V.38 - Caixa de Diálogo Exemplos de Entrada e Saída

A fim de que possamos armazenar e/ou eliminar um ou mais exemplos de entrada e saída na ferramenta, temos que criar a Caixa de Diálogo Exemplos de Entrada e Saída pressionando-se o botão direito do *"mouse"* em qualquer posição dentro da área do cliente da janela Editor de Entrada ou da janela Editor de Saída.

A Caixa de Diálogo Exemplos de Entrada e Saída que pode ser vista na figura, V.38 apresenta dois botões:

- *O botão "→ 0"* - armazena simultaneamente o exemplo de entrada e o de saída desejados e que estão representados respectivamente nas janelas Editor de Entrada e Editor de Saída no momento do armazenamento.
  Permite que vários exemplos sejam armazenados, sendo que tais exemplos serão utilizados ciclicamente pelo algoritmo de aprendizagem.
- *O botão "X 0"* - retira do conjunto de exemplos armazenados pela ferramenta, o par de exemplos de entrada e de saída que estão representados pelas janelas Editor de Entrada e Editor de Saída no momento em que este botão é pressionado.

Sendo armazenado pelo menos um exemplo de entrada/saída na ferramenta, o exemplo padrão deixa de existir para a mesma. Se forem eliminados todos os exemplos armazenados, o exemplo padrão volta a ser o único exemplo utilizado.

V.5.2 - Visualização dos Exemplos Armazenados.

Quando a ferramenta apresenta exemplos armazenados em seu arquivo de armazenamento, os mesmos serão mostrados ciclicamente nas janelas Editor de Entrada e Editor de Saída à medida que forem transcorrendo as iterações do algoritmo de aprendizagem.

Na figura V.39, podemos ver como três exemplos de entrada/saída são apresentados ciclicamente durante a aprendizagem.

V.5.3 - Aprendizado Seqüencial e Aprendizado Simultâneo.

O Sirena permite dois tipos de aprendizado supervisionado, o aprendizado seqüencial e o aprendizado simultâneo que diferem entre si pela forma que os exemplos de entrada/saída são apresentados ao algoritmo de aprendizagem.

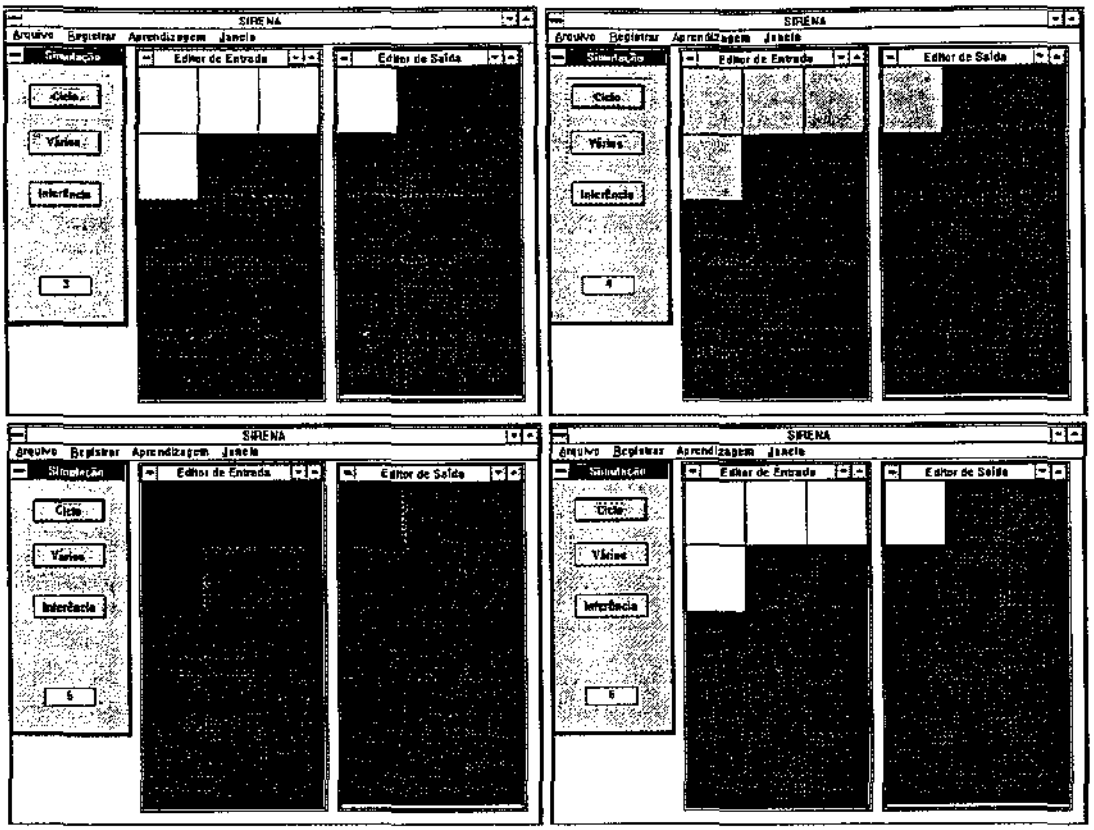


Figura V.39 - Apresentação cíclica dos exemplos de entrada e saída armazenados.

O aprendizado seqüencial, isto é, aquele onde cada exemplo é "aprendido" pela rede antes que um novo exemplo seja apresentado, é feito através da seguinte seqüência:

- Armazenamento de um exemplo de entrada/saída pelo botão "→" da Caixa de Diálogo Exemplos de Entrada e Saída.
- Aprendizado do exemplo armazenado através dos botões "Ciclo" ou "Vários" da Caixa de Diálogo Principal explicada na próxima seção.
- Retirada do exemplo armazenado através do botão "X 0" da Caixa de Diálogo Exemplos de Entrada e Saída.
- Repetição do processo com outro exemplo de entrada/saída.

O aprendizado simultâneo, isto é, aquele em que todos os exemplos são "aprendidos" ao mesmo tempo segue uma seqüência diferente:

- Armazenamento de cada exemplo de entrada/saída pelo botão "→" da Caixa de Diálogo "Exemplos de Entrada e Saída".
- Aprendizado dos exemplos armazenados através do botão "Ciclo" ou do botão "Vários" da Caixa de Diálogo Principal.

V.5.4 - Execução da Simulação e Inferência.

Para que se execute o processo de aprendizagem e o de inferência é necessário o uso da Caixa de Diálogo Principal, que pode ser vista na figura V.40

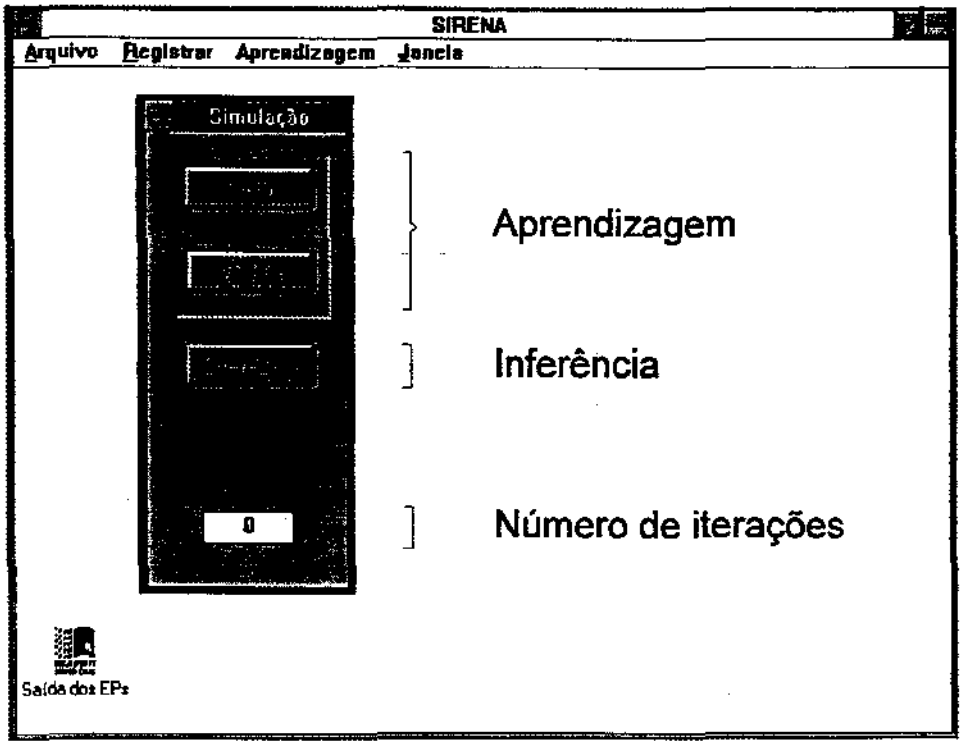


Figura V.40 - Caixa de Diálogo Principal.

Esta caixa que apresenta dois grupos de botões e um visor de iterações é apresentada sempre no ato de criação de qualquer janela da ferramenta.

O primeiro grupo, responsável pela aprendizagem da rede, constitui-se de dois botões:

- *O Botão de Aprendizagem "Ciclo"* - realiza uma iteração do algoritmo de aprendizagem. É utilizado principalmente quando se quer examinar as mudanças que uma iteração da aprendizagem causa na rede.
- *O Botão de Aprendizagem "Vários"* - permite que se realize um número, definido pelo usuário, de ciclos do algoritmo de aprendizagem. Seu principal uso é o de treinar a rede para realizar as tarefas desejadas não se preocupando em examinar alterações do estado da rede passo-a-passo, o que é feito pelo botão "Ciclo".

  A aprendizagem com o botão "Vários" acontece, do ponto de vista do usuário, como se fosse realizado apenas um ciclo, pois as modificações dos conteúdos das janelas do Sirena só poderão ser notadas após a realização do último ciclo do algoritmo de aprendizagem.

Ao pressionarmos o botão "Vários" estaremos automaticamente criando a **Caixa de Diálogo Várias Iterações** que pode ser vista na figura V.41.

Esta caixa apresenta a janela de edição de texto "Número de Iterações:" que permite a entrada do número de iterações ou ciclos desejados para o algoritmo de aprendizagem. Tal número para ser válido precisa ser maior que zero.

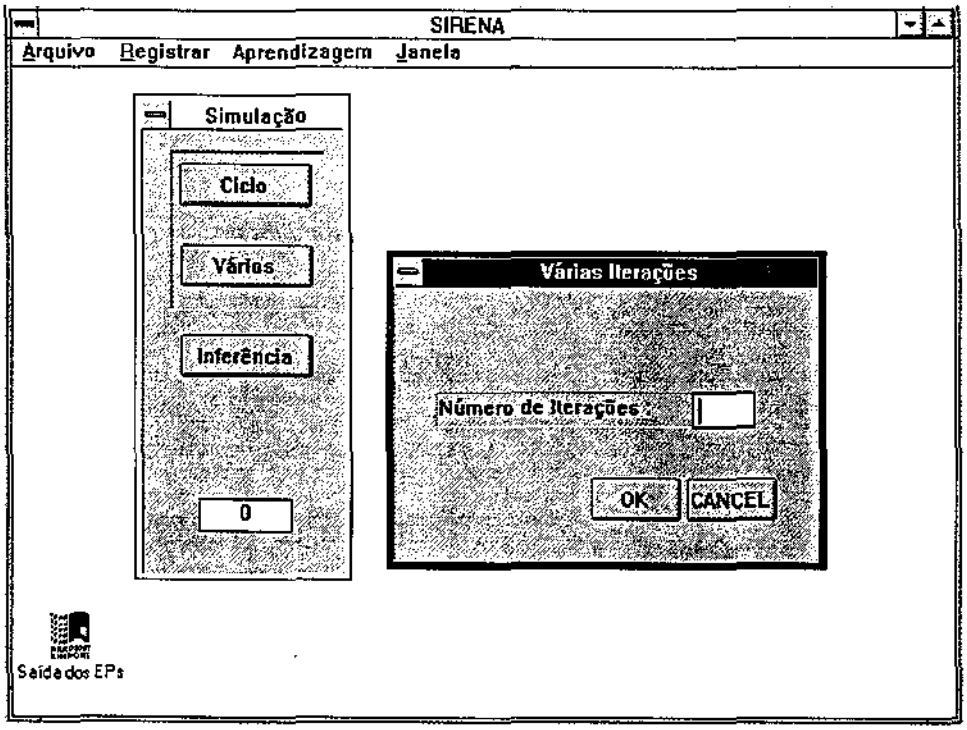


Figura V.41 - Caixa de Diálogo Várias Iterações.

O segundo grupo da Caixa de Diálogo Principal é responsável pelo processo de inferência da rede, ele é composto por um único botão chamado de "Inferência".

Encerrado o processo de aprendizagem, o usuário utiliza a rede para obter informações de saída a partir de informações de entrada fornecidas a ela.

O usuário introduz suas informações de entrada através da janela Editor de Entrada, pressiona o botão "Inferência" da Caixa de Diálogo Principal, que não provocará mudança nos valores dos pesos da rede, e observa os resultados obtidos através da janela Saída Estruturada.

O último grupo desta caixa é representado pelo Visor de Iterações que apresenta qual é a iteração atual do simulador.

Apertando-se o botão "Ciclo" ou o botão "Inferência", o Visor de Iterações aumentará em um o número por ele apresentado, já com o botão "Vários", o visor aumentará o correspondente ao número introduzido na Caixa de Diálogo "Várias Iterações".

Quando ocorrer um retrocesso de iterações como será explicado na próxima seção, o Visor de Iterações indicará o valor da iteração a qual houve o retrocesso.

V.5.5 - Armazenamento e Recuperação de Simulação/Inferência.

Muitas vezes o usuário da ferramenta tem a necessidade de armazenar os processos de aprendizagem e inferência de uma RNA, seja para poder observar o comportamento global da rede durante esses processos, seja para poder retroceder a um estado específico do simulador durante uma determinada iteração ou ainda para armazenar a topologia e exemplos de entrada e saída da rede.

Para permitir essa facilidade criou-se um mecanismo de armazenamento e recuperação de simulação e/ou inferência que pode ser acionado através do item Registrar do *"menu"* principal.

Ao ser acionado, ele automaticamente cria a **Caixa de Diálogo Registro da Simulação** que pode ser vista na figura V.42.

Essa caixa foi dividida em dois grupos: o primeiro é responsável pela gravação e recuperação das informações da rede e apresenta a opção "Gravar Execução" e a janela de edição de texto "Ir à iteração num" sendo este grupo .

O segundo grupo é formado apenas pela opção "Arquivo não padrão" responsável pela escolha do arquivo para onde as informações serão armazenadas e de onde serão recuperadas.

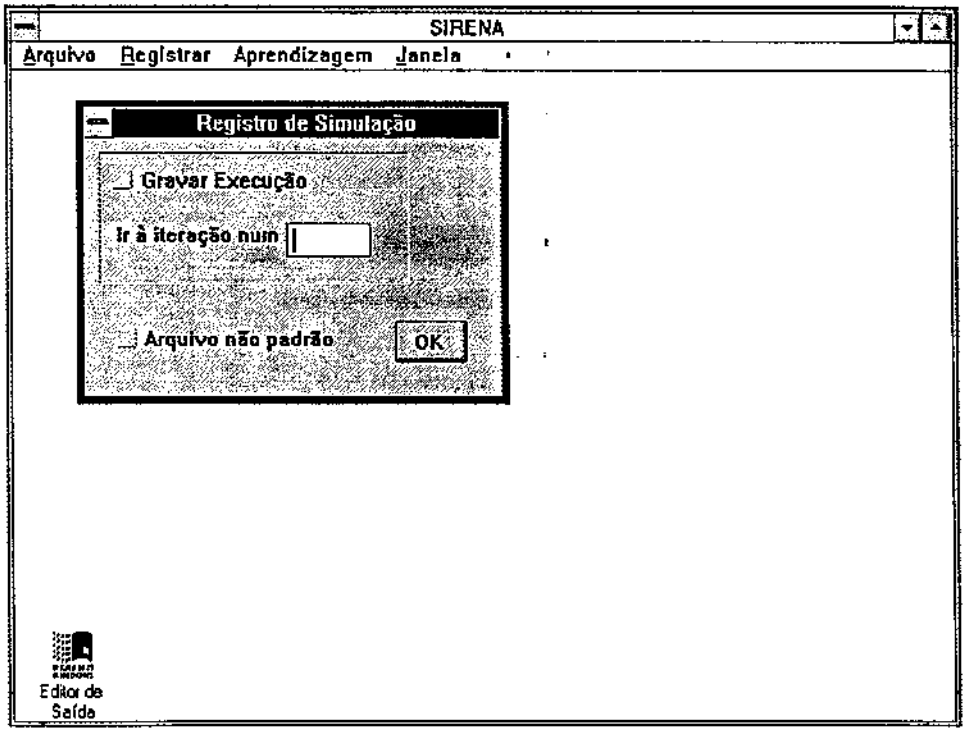


Figura V.42 - Caixa de Diálogo Registro da Simulação.

V.5.6 - Armazenamento de Informações.

Alterando-se o estado da opção "Gravar Execução" sem, porém, alterar o estado do "Arquivo não padrão", estaremos demonstrando o desejo de armazenar a aprendizagem e/ou

inferência da rede no arquivo padrão, isto é, um arquivo externo denominado "LogFile.log" e criado especificamente para o armazenamento de informações da rede.

A gravação da simulação se dará a partir da iteração corrente do simulador que pode ser zero se for no início da simulação ou outro número se a simulação já tiver sido iniciada. O número da iteração corrente pode ser vista na Caixa de Diálogo Principal.

O usuário, portanto, não é obrigado a armazenar a simulação inteira mas somente a parte da mesma que lhe interessa.

A partir do momento em que o botão de OK foi pressionado, a caixa é destruída, e a gravação se dá de forma transparente ao usuário, bastando que ele aperte os botões da Caixa de Diálogo Principal.

Para que a gravação da simulação não corrompa o sistema, o que ocorre quando o arquivo possuir um tamanho excessivamente grande devido a muitas iterações, o arquivo é limitado na atual implementação a um tamanho máximo de 1 Mbytes.

Ultrapassado esse limite, a ferramenta parará a simulação ou inferência e a gravação, avisando ao usuário sobre o acontecido através de uma caixa de diálogo mostrada na figura V.43

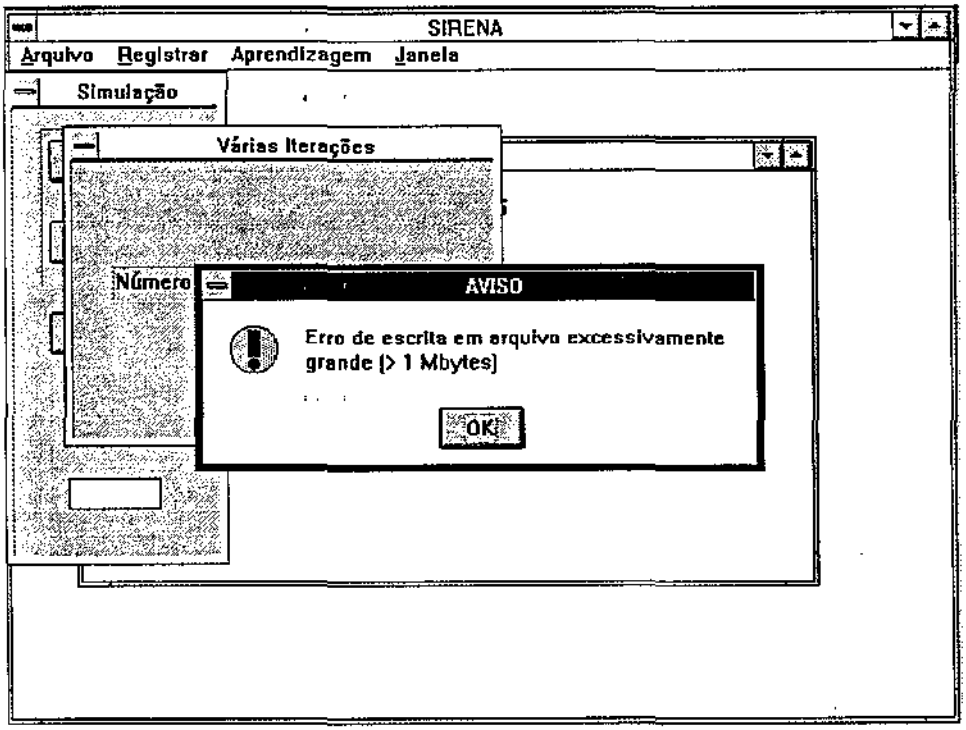


Figura V.43 - Aviso de arquivo muito extenso.

Um arquivo não padrão é qualquer arquivo cujo nome seja diferente de "LogFile.log" e que se destine como o arquivo padrão ao armazenamento da simulação/inferência.

Seus objetivos são:

- Permitir, caso haja espaço disponível, a continuação de uma simulação/inferência cujo processo tenha sido interrompido devido ao tamanho do arquivo padrão ser excessivamente grande.
- Permitir a gravação de várias simulações/inferências, sendo que cada uma em um arquivo não padrão diferente.

A seleção de um arquivo não padrão se faz acionando-se a opção "Arquivo não padrão" da Caixa de Diálogo Registro da Simulação. Automaticamente a **Caixa de Diálogo Arquivo Não Padrão** é apresentada permitindo ao usuário a escolha do nome de novo arquivo não padrão (vide figura V.44).

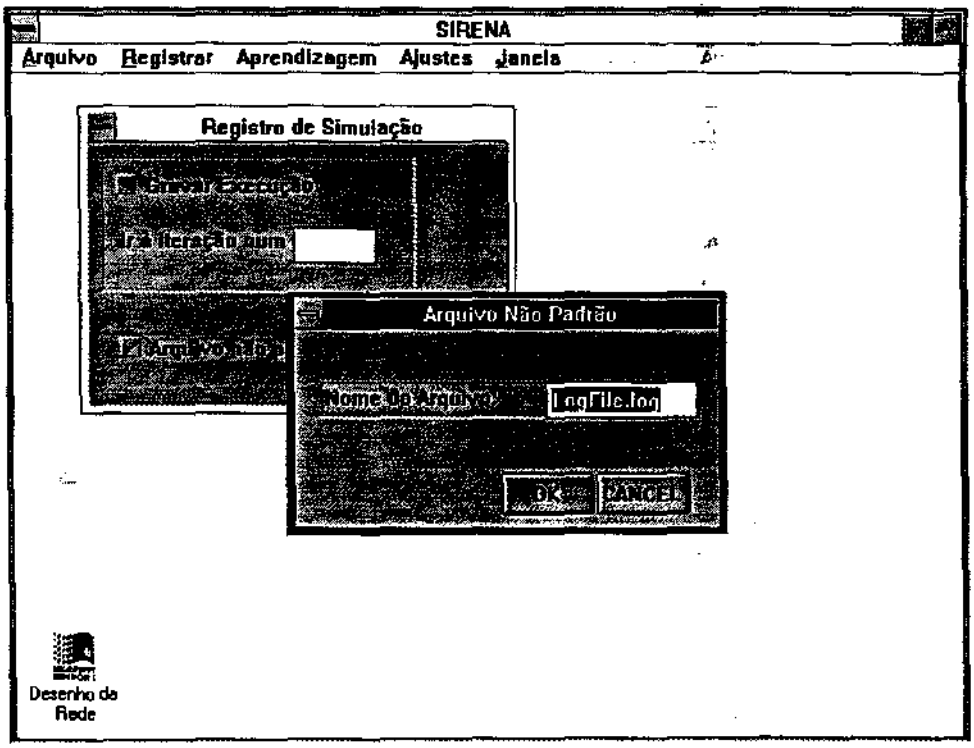


Figura V.44 - Caixa de Diálogo Arquivo Não Padrão.

Digitando um nome qualquer para o arquivo não padrão e apertando o botão de OK, toda informação a ser armazenada será direcionada a este novo arquivo.

A opção "Arquivo não padrão" ficará marcada indicando o uso de um arquivo não padrão, mesmo em chamadas posteriores à Caixa de Diálogo Registro da Simulação. A opção "Gravar Execução" não estará marcada após a destruição da Caixa de Diálogo Arquivo Não Padrão indicando ao usuário que todo o processo de gravação não está mais acionado.

Após a mudança de arquivo, para que efetivamente se grave no arquivo recém selecionado é necessário marcar a opção "Gravar Execução" indicando o desejo de gravar a simulação e/ou inferência.

Se ao invés do botão OK, for pressionado o botão CANCEL na Caixa de Diálogo Arquivo Não Padrão, o arquivo de armazenamento voltará a ser o padrão, retornando o estado anterior da Caixa de Diálogo Registro da Simulação, isto é, a opção "Arquivo não padrão" será apagada indicando o uso do arquivo padrão e a opção "Gravar Execução" se manterá da mesma forma que estava antes da chamada da Caixa de Diálogo Arquivo Não Padrão.

V.5.7 - Recuperação das Iterações Passadas.

Supondo-se a existência de um arquivo de informações não vazio, podemos recuperar uma iteração especifica do simulador através da caixa de edição de texto "Ir à iteração num" da **Caixa de Diálogo Registro da Simulação**. Indicada uma iteração desejada e apertado o botão OK, o simulador assumirá o mesmo estado que se encontrava em tal iteração.

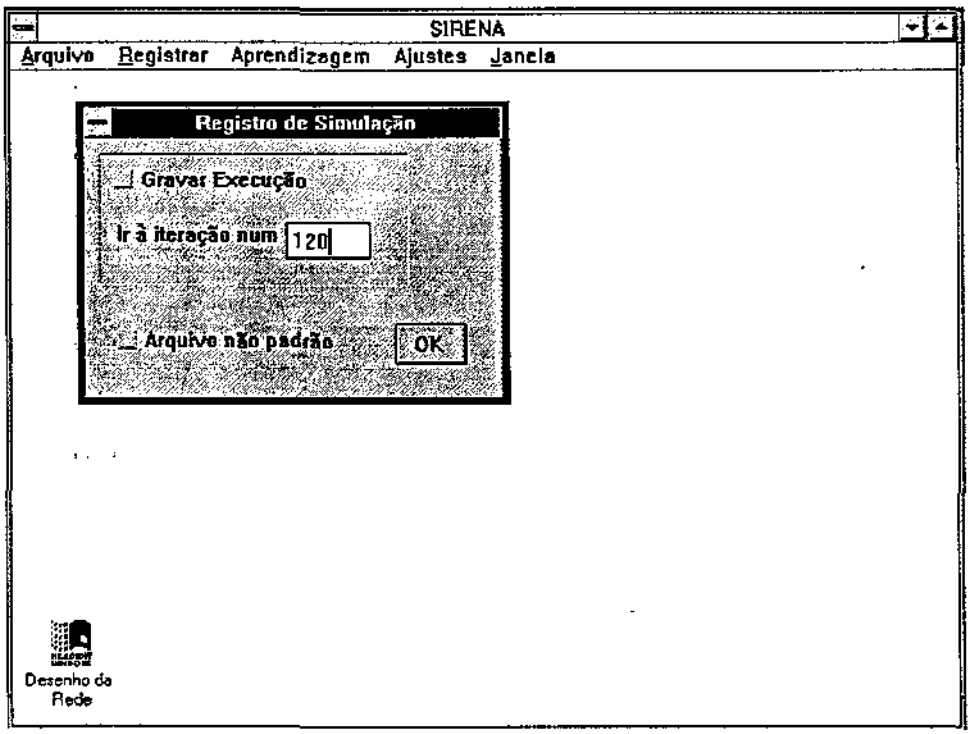


Figura V.45 - Recuperação de iteração passada.

Nesta operação de recuperação de uma iteração anterior, o estado da opção "Gravar Execução" não é considerado, mas o do "Arquivo não padrão" sim, pois somente através dele é que se sabe se o arquivo de armazenamento do qual serão recuperadas as informações é ou não o arquivo padrão.

Se a recuperação de informações de um arquivo não padrão for feita logo após a gravação dessas informações, o nome do arquivo não padrão já estará armazenado, mas se for feita em uma outra ocasião de utilização da ferramenta, o nome do arquivo terá de ser fornecido novamente através da Caixa de Diálogo Arquivo Não Padrão.

Estão previstas mensagens de erro, através de caixas de diálogo, para situações como: tentativa de leitura em arquivo vazio, arquivo não existente, iteração não encontrada, etc.

V.5.8 - Utilização Não Trivial do Mecanismo de Armazenamento e Recuperação de Informações.

Toda vez que selecionarmos a opção "Gravar Execução", estaremos de alguma forma alterando o arquivo de armazenamento.

Se selecionarmos essa opção sem termos feito um retrocesso a uma iteração anterior através da janela "Ir à iteração num" estaremos dizendo à ferramenta que desejamos gravar e que não utilizamos antes o arquivo de armazenamento e portanto, qualquer coisa nele armazenada é descartável.

Desta forma o arquivo de armazenamento é criado, se ele não existir, ou apagado se ele já existir. De uma forma ou de outra, estaremos começando a gravação em um arquivo sem informação gravada em seu interior.

Se selecionarmos a opção de gravação após ter sido feito um retrocesso a uma iteração anterior então duas situações podem ocorrer:

- *A gravação é feita logo após o retrocesso* - neste caso o arquivo de armazenamento é apagado a partir da posição correspondente à iteração retrocedida, pois estamos dizendo que gostaríamos de gravar a simulação a partir daquele ponto e tudo o que já está gravado dali para frente não mais nos interessa.
- *A gravação é feita após o retrocesso, mas depois de algumas iterações do simulador* - este caso ocorre quando retrocedemos a alguma iteração anterior, avançamos algumas iterações para frente através dos botões "Ciclo" ou "Vários" ou "Inferências" e só depois selecionamos a opção de gravação.
  Neste caso a ferramenta trunca o arquivo após o retrocesso e volta a gravar a partir da iteração onde a opção "Gravar Execução" é selecionada. O arquivo, portanto, fica sem as iterações que ocorreram entre esses dois eventos.

Se a opção de gravação estiver selecionada e resolvermos então fazer um retrocesso, após o mesmo, a opção será apagada deixando a opção para o usuário de deixar o arquivo intacto ou de truncá-lo caso se selecione o mesmo.

Mesmo estando a opção "Gravar Execução" selecionada, ao retornarmos da Caixa de Diálogo Arquivo Não Padrão onde indicamos o nome do arquivo não padrão, a opção estará apagada deixando a possibilidade ao usuário de manter o conteúdo do arquivo intacto, caso ele já exista, apagá-lo a partir de uma posição a qual houve retrocesso ou ainda criá-lo caso ele não exista.

Todas estas operações são feitas utilizando-se os mecanismos explicados anteriormente.

V.5.9 - Ajuste do Algoritmo de Aprendizagem.

Quando selecionamos o item "Aprendizagem" do *"menu"*, criamos a **Caixa de Diálogo Algoritmo de Aprendizagem** que é responsável pelo ajuste dos parâmetros Alfa e Eta que alteram a eficiência do algoritmo de "Backpropagation" de acordo com a topologia da rede e as informações de treinamento.

Essa caixa como pode ser vista na figura V.46 , é composta basicamente por duas janelas de edição de texto, respectivamente denominadas "Alfa" e "Eta", e que permitem a entrada dos valores correspondentes a esses parâmetros.

Quando apresentada, a Caixa de Diálogo Algoritmo de Aprendizagem mostra em suas janelas os valores anteriores dos parâmetros, sendo que os valores padrões para os mesmos são iguais a 0,5.

Em um uso típico do Sirena, são necessárias pelo menos quatro janelas ativas: para a entrada de informações na rede, as janelas Editor de Entrada e Editor de Saída; para a criação e alteração da topologia da rede, a janela Desenho da Rede e finalmente para a verificação dos resultados de saída da mesma, a janela Saída Estruturada ou a janela Valores de Saída da Rede.

Podemos, portanto, ter a operacionalidade do Sirena utilizando-se somente suas janelas gráficas, que não por coincidência se destinam a permitir uma visão qualitativa das informações da rede, isto é, uma visão com um nível menos detalhista destas informações.

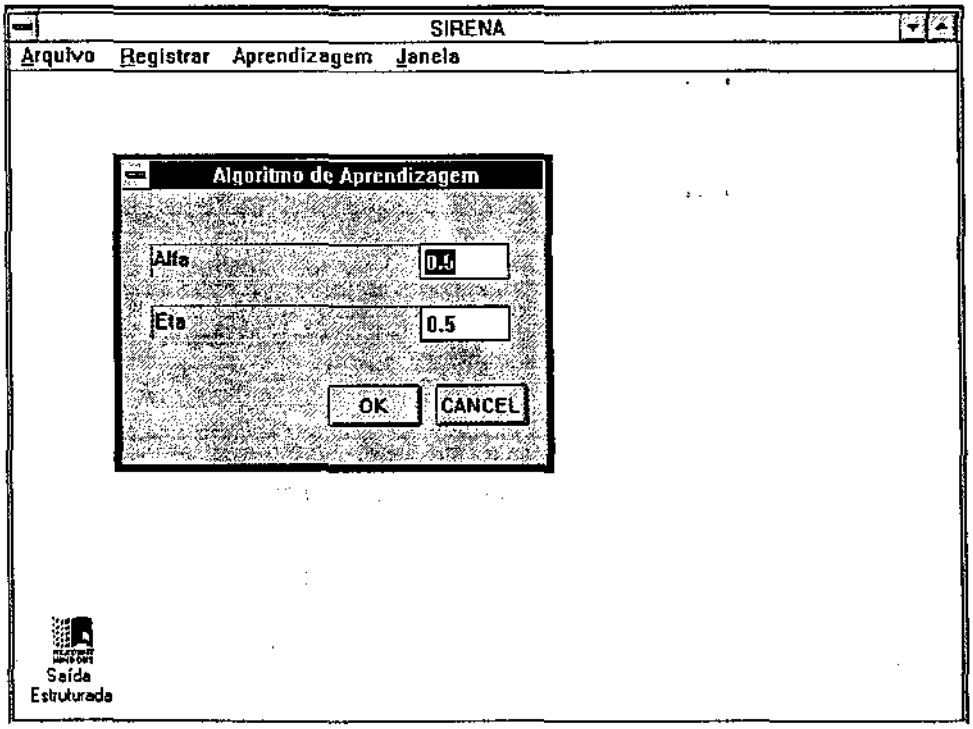


Figura V.46 - Caixa de Diálogo Algoritmo de Aprendizagem.

Quando precisarmos, porém, de informações mais completas sobre a rede do que as fornecidas pelas janelas gráficas, devemos recorrer às janelas textuais que visam formar um modelo mais aperfeiçoado do funcionamento da rede fornecendo informações mais específicas (visão quantitativa).

O fornecimento mais completo de informações por parte do Sirena se dará com a combinação das informações quantitativas (janelas textuais) com as informações qualitativas (janelas gráficas).

O aspecto importante é que é deixado ao usuário a opção de configuração das representações que deseja. Com isso não se impõe nenhum critério *a priori* do que é ou não importante ser observado.

V.6 - Considerações Finais.

Podemos descrever a interface gráfica do Sirena através do seu *"menu"* principal que resume as três principais atividades da ferramenta; a aprendizagem da rede (item aprendizagem), o registro dos processos de inferência e aprendizagem (item registrar) e a manipulação da topologia da rede (item arquivo).

A aprendizagem da rede representada aqui por seu algoritmo de aprendizagem pode ser alterada com uma única caixa de diálogo que permite o ajuste dos parâmetros do algoritmo de "Backpropagation". Em uma implementação posterior poderiam constar neste item opções para a escolha de outros algoritmos e seus respectivos parâmetros.

O registro das informações da rede e o retorno a uma iteração armazenada podem ser feitos através de caixa de diálogo que permite a escolha por um tipo de utilização facilitada ou por uma utilização mais poderosa, o mecanismo de recuperação não trivial.

A manipulação da topologia da rede é feita através do item arquivo que permite a criação de dois tipos de janelas, as textuais que apresentam informações numéricas (quantitativas) e as janelas gráficas que se destinam a lidar com as informações de entrada e saída e com a rede propriamente dita, utilizando-se de uma transmissão de informações não numéricas (qualitativas).

A conceituação da interface gráfica da ferramenta pode ser vista na figura V.47

Os itens Arquivo e Aprendizagem se destinam a fornecer informações e dar a liberdade de escolha e manipulação dos elementos envolvendo as RNAs qualquer que seja o estágio do processo de aprendizagem ou de simulação que está sendo executado.

O item Registrar se destina a permitir o cruzamento de informações e a possibilidade de recuperação de um estado anterior da ferramenta quando a liberdade de escolha e manipulação não conduzir a resultados satisfatórios.

É importante salientar que somente com o uso intenso do Sirena poderão ser feitas todas as depurações necessárias, algumas das dificuldades na utilização dos recursos já puderam, porém, ser detectadas com a atual implementação.

Uma questão interessante a ser levantada é a utilização das cores para a associação com intervalos numéricos.

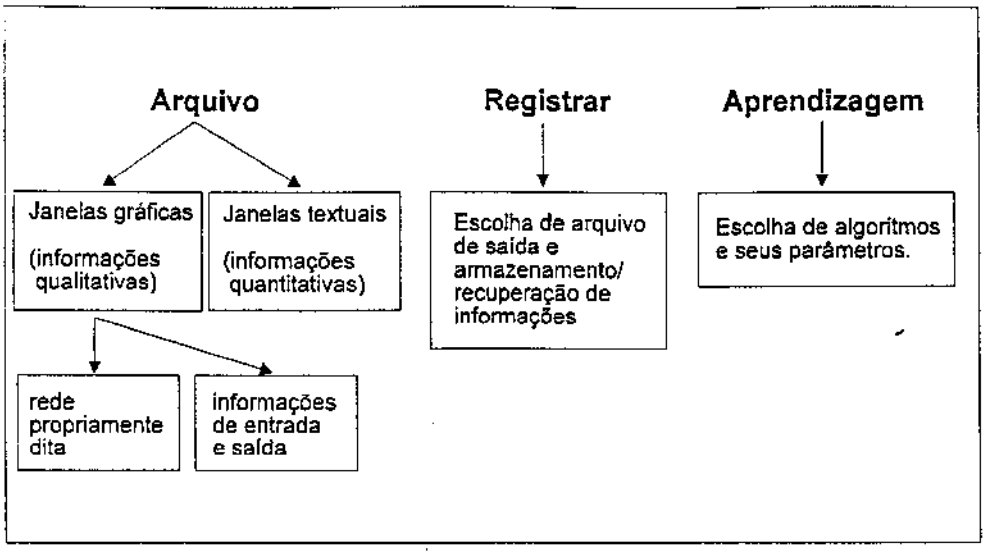


Figura V.47 - Conceituação da interface gráfica.

Quando o número de intervalos é pequeno, a associação cor/intervalo pode ser bem sucedida com a prática de uso da ferramenta, mas no caso de RNAs, poucos intervalos podem significar muitas iterações do algoritmo de aprendizagem para se mudar de um intervalo para outro, o que dificulta o acompanhamento em mais detalhes da evolução da rede.

Um número maior de intervalos, portanto, é desejável, mas dificulta a sua associação por parte do usuário, que teria que ficar constantemente verificando uma tabela de conversão cor/intervalo. Esta conversão fica ainda mais prejudicada pelo fato dos monitores atuais dos microcomputadores permitirem um número limitado de "cores puras", isto é, quando em um espaço do monitor conseguimos enxergar apenas uma cor e não um conjunto de "*pixels*" com cores variadas, dificultando a distinção entre as cores apresentadas.

A utilização da cor para indicar apenas a mudança de intervalo numérico sem a preocupação com os limites numéricos deste intervalo, parece ser uma solução aceitável para o problema.

Outra questão a ser pensada a respeito da utilização da interface é a escolha entre a facilidade de entendimento da ferramenta com o uso de um item a mais no *"menu"* e a rapidez de acesso a esse recurso com a utilização do segundo botão do *"mouse"*.

A escolha pela segunda opção nas janelas Editor de Entrada, Editor de Saída e Desenho da Rede mostrou-nos que usuários sem conhecimento prévio do funcionamento da ferramenta encontram maior dificuldade em sua utilização, enquanto que aqueles com familiaridade com seu funcionamento acham a opção menos desgastante que o acionamento do mesmo recurso através do *"menu"*.

Para finalizar, outro problema observado que merece atenção é a possível dificuldade de associação por parte do usuário entre as informações de entrada e saída quando imaginadas como um padrão bidimensional, por exemplo o formato de uma letra, e os EPs das camadas de entrada e saída que estão dispostas linearmente. Esta dificuldade fica clara quando queremos cruzar informações entre as janelas Saída Estruturada e Valores de Saída de Rede ou entre as janelas Editor de Entrada e Valores de Entrada da Rede.

Capítulo VI

Descrição da Implementação da Ferramenta

Neste capítulo mostraremos características de implementação do sistema Sirena tais como a plataforma utilizada em seu desenvolvimento, a linguagem de programação escolhida, e a sua organização em blocos conceituais e módulos de programação.

VI.1 - Plataforma e Linguagem.

A plataforma utilizada para a implementação do sistema Sirena foi um microcomputador com processador Intel 486 DX possuindo freqüência de 33 MHz, memória de 8 Mbytes e monitor colorido de 14 polegadas SVGA.

O ambiente operacional escolhido foi o Microsoft Windows, versão 3.1, devido a seu reconhecido sucesso em manipulação de janelas e outros recursos gráficos e também por sua grande disseminação entre os microcomputadores da "linha PC".

A linguagem utilizada para a implementação do sistema foi a C++, escolhida por sua compatibilidade com o ambiente Windows, por sua adequada velocidade de execução, seu reconhecido poder de programação e também por seu paradigma de orientação a objetos utilizado nesse projeto para a representação dos elementos de processamento e da Rede Neural Artificial (RNA) propriamente dita.

VI.2 - Estrutura da Implementação.

O objetivo do sistema implementado é construir uma ferramenta que possibilite o entendimento e a visualização dos dois processos fundamentais pelos quais toda RNA está sujeita: a simulação e a aprendizagem.

- *Simulação* - é um processo composto por dois subprocessos:

⇒ *Definição da topologia* - caracteriza-se pela definição, por parte de projetista, do número de elementos de processamento (EPs) que a rede vai possuir, quais funções

que estarão associadas a cada EP, como os EPs estarão distribuídos em camadas e como os mesmos se interligarão para formar a rede.

⇒ *Inferência* - é o objetivo final de uma RNA, pois é o subprocesso onde fornecidos os valores de entrada da rede, consegue-se inferir, ou seja, obter através de sua topologia, os valores de saída da mesma. Os resultados obtidos são dependentes das informações armazenadas nas ligações da rede.

- *Aprendizagem* - uma vez definida a topologia da rede, sua inferência não produzirá resultados significativos se a rede não for treinada para produzir os resultados desejados. Esse processo de treinamento da rede é que chamamos de aprendizagem e que consiste basicamente na alteração dos valores dos limiares e pesos das ligações de forma a obter o desempenho desejado da rede.

Para organizar a implementação, criou-se o conceito de *bloco conceitual* que tem o papel de representar, a nível conceitual, algumas atividades gerais da ferramenta tais como a simulação, a aprendizagem, o armazenamento de informações e a visualização da rede.

Cada bloco conceitual agrupa dentro de si *módulos de programação* que implementam, a nível de código, as várias atividades por ele representadas.

A ferramenta implementada possui dois blocos conceituais denominados Simulação e Visualização, sendo que sua organização pode ser vista na figura VI.1.

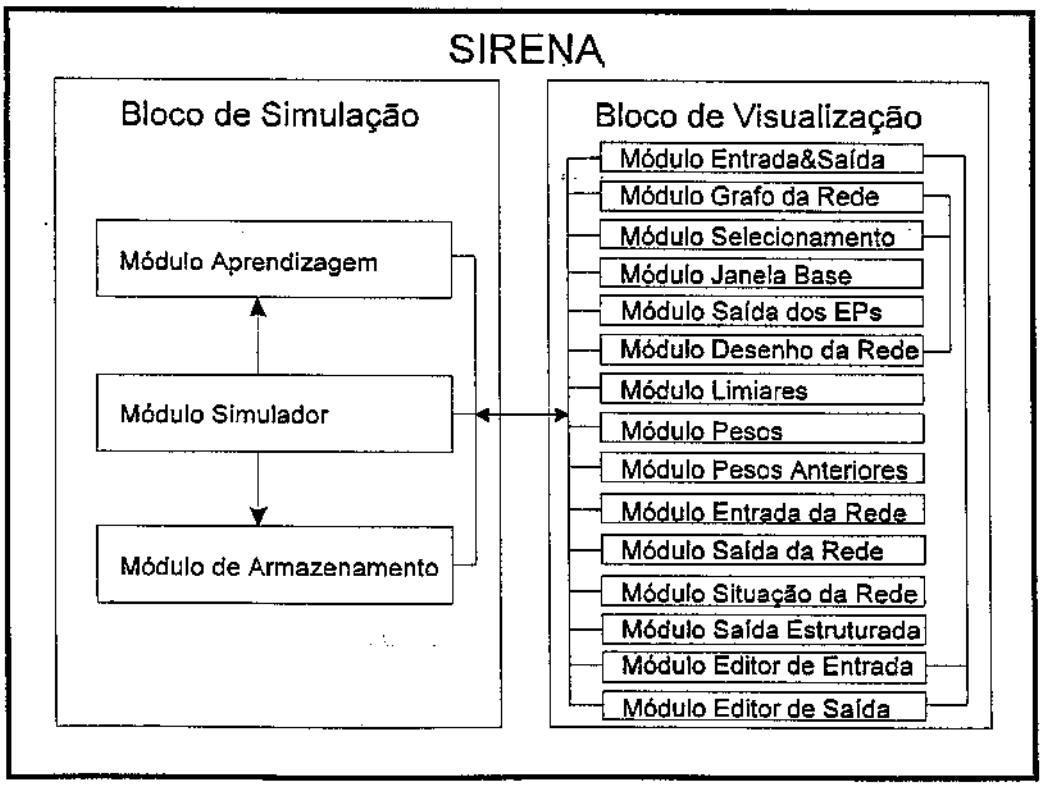


Figura VI.1 - Divisão da ferramenta em Blocos Conceituais e Módulos de Programação.

O Bloco de Simulação representa os processos de simulação e aprendizagem de uma RNA, sendo composto por três módulos de programação:

- *Módulo Simulador* - implementa a atividade de definição da topologia da rede, ou seja, o processo de criação e alteração da topologia e a atividade de inferência da mesma.
- *Módulo de Aprendizagem* - responsável pelo processo de aprendizagem da rede, foi definido à parte do Módulo Simulador objetivando facilitar a implementação de vários algoritmos de aprendizagem.
  Na implementação descrita neste trabalho implementou-se o algoritmo de "Backpropagation" escolhido como protótipo por ser um dos mais referenciados na literatura e por possuir a maioria das características significativas presentes em outros algoritmos de aprendizagem.
- *Módulo de Armazenamento* - tem como objetivo permitir o armazenamento em um arquivo externo de todo o processo de aprendizagem e/ou inferência da rede, para que possam ser recuperados a qualquer momento conforme a necessidade do usuário.

O Bloco de Visualização é responsável pela implementação do objetivo de facilitar o entendimento de uma RNA.

Para isso faz uso de recursos gráficos que interagindo com o Bloco da Simulação, permitem a construção, alteração, visualização do estado de uma RNA, assim como o acompanhamento interativo dos seus processos de aprendizagem e inferência

A cada janela implementada da interface gráfica do Sirena, está associado pelo menos um módulo de programação do Bloco de Visualização.

Explicaremos a seguir com maiores detalhes a estrutura dos blocos de Simulação e Visualização.

VI.3 - Bloco de Simulação.

Esse bloco representa o conjunto das atividades essenciais que se espera de um simulador, independente de sua interface de interação.

As atividades que são realizadas pelos Módulos Simulador, Aprendizagem e Armazenamento começam com a fase de implementação de uma RNA partindo-se do projeto inicial da mesma, passam pelo processo de treinamento e finalizam com a inferência da rede a partir de informações de entrada. Estas duas últimas fases podem ou não serem armazenadas em um arquivo externo.

VI.3.1 - Módulo Simulador.

O Módulo Simulador é o responsável pela implementação de três atividades:

- *Criação da rede* - feita através da criação de elementos de processamento ou EPs com suas respectivas funções de somatório e transição, distribuição dos EPs em camadas e a interconexão dos mesmos com a criação de ligações e seus respectivos pesos .
- *Alteração da rede* - onde a topologia da RNA é alterada através das seguintes atividades:

  ⇒ Acréscimo ou diminuição do número de EPs existentes na rede.

  ⇒ Criação e/ou eliminação de ligações entre os EPs.

  ⇒ Alterações dos valores dos pesos das ligações.

  ⇒ Alterações dos limiares associados aos EPs da rede.
- *Inferência* - de posse de um vetor de entrada fornecido pelo usuário, vetor que associa a cada EP da camada de entrada um valor inicial para o mesmo, o Módulo Simulador processa esses valores através das unidades da rede obtendo um vetor de saída, possuidor dos valores finais de cada EP da camada de saída da rede.

O Módulo Simulador é constituído basicamente de duas classes :

- *A classe Node* - responsável pela representação do elemento de processamento da rede e de suas ligações.
- *A classe Net* - responsável pela organização dos objetos criados pela classe Node em camadas, assim como pelo armazenamento dos valores de entrada e saída desejados para a aprendizagem da RNA.

VI.3.1.1 - A Classe Node.

A classe Node tem o objetivo de representar um elemento de processamento da RNA e todas as ligações que saem deste EP, possuindo para tal as seguintes estruturas de dados:

```
float Output;
float Threshold;
float Delta;
float Weights[MAXCOLUMM];
float LastWeights[MAXCOLUMM];
```

Onde:

Output - representa o valor de saída do EP (valor da função de transferência).

Threshold - representa o valor do limiar do EP. Durante a fase de aprendizagem o valor do limiar é ajustado imaginando-se que o mesmo seja o peso de uma ligação que une um EP fictício cujo valor de saída é constante e igual a -1 ao EP que armazena o limiar.
Sendo considerado apenas como mais um peso, o limiar é ajustado da mesma forma que o outros pesos da rede.

Delta - no Módulo Simulador esta variável pode ser utilizada como uma variável extra para armazenar um número real, o Módulo de Backpropagation a utiliza, porém, para armazenar no EP o termo de erro, também conhecido como delta, do algoritmo de "Backpropagation".

Weights[MAXCOLUMM] - é o vetor que armazena os valores dos pesos das ligações do EP considerado com os EPs da camada seguinte que são em número de MAXCOLUMM.
O valor Weights[i], portanto, representa o peso da ligação do EP considerado com o outro EP da camada seguinte na posição *i* desta camada.

LastWeights[MAXCOLUMM] - o objetivo de um algoritmo de aprendizagem é o de alterar os valores dos pesos das ligações de forma a aumentar a eficiência da rede.
A cada iteração deste algoritmo espera-se que várias das ligações da rede tenham seus pesos alterados.
O vetor LastWeights armazena os valores dos pesos das ligações que saem de um determinado EP antes que os mesmos sejam alterados por mais uma iteração do algoritmo.
Podemos então considerar o vetor LastWeights como aquele que contém os valores dos pesos de um EP da iteração anterior do algoritmo de aprendizagem.

VI.3.1.2 - A Classe Net.

Vimos que a classe Node representa um EP e suas respectivas ligações com EPs da camada seguinte.

É, porém, de responsabilidade da classe Net a criação dos objetos da classe Node e a organização dos mesmos em camadas.

Cabe a ela também o armazenamento dos valores de entrada e saída desejados para a rede utilizados no aprendizado supervisionado, assim como informações sobre a distribuição dos EPs por camada e o número de iterações já realizadas pelo simulador.

Para tanto, a classe Net apresenta as seguintes estruturas:

```
typedef Node NeuralNetwork[MAXROW][MAXCOLUMM];
typedef NeuralNetwork FAR * Neural;
Neural NeuralNet;

float   Input[MAXCOLUMM];
float   Output[MAXCOLUMM];
int     Columm[MAXROW];
int     HiddenNumber;
int     Inp,Out;
long    Iteration;
```

Onde:

NeuralNet - é um apontador para a matriz NeuralNetwork de objetos Node, onde as linhas da mesma representam as camadas da rede e as colunas, a posição do EP dentro da camada. Essa matriz possui o tamanho de MAXROW linhas e MAXCOLUMM colunas, o que na atual implementação correspondem a 10 camadas com 15 EPs por camada, totalizando 150 EPs.

Input[MAXCOLUMM] - armazena através de um vetor os valores de entrada da rede utilizados tanto para a fase de aprendizagem como para a fase de inferência.

Output[MAXCOLUMM] - armazena os valores de saída desejados para a rede. Tais valores serão utilizados conjuntamente com os valores do vetor Input no aprendizado supervisionado, isto é, aquele onde os valores de entrada e de saída desejados são fornecidos à rede.

Columm[MAXROW] - armazena em cada uma das suas posições o número de EPs criados na camada correspondente da rede, isto é, o valor de Columm[i] indica quantos EPs existem na camada *i* da rede.

HiddenNumber - indica qual é o número de camadas ocultas presentes na rede.

Inp, Out - são variáveis booleanas que indicam respectivamente a presença quando seu valor é 1, ou ausência quando seu valor é 0, das camadas de entrada e de saída da rede. Em uma rede que só tenha uma camada , consideramos essa camada como sendo a camada de entrada (Inp = 1, HiddenNumber = 0, Out = 0).
Podemos calcular o número de camadas da rede através da operação:

$$Inp + HiddenNumber + Out$$

e o número de EPs da rede através de:

$$\sum_i Columm[i].$$

Iteration - armazena o número de vezes em que o simulador, a partir de informações de entrada, obteve informações de saída da rede.
A variável **Iteration**, portanto, indica o número de iterações realizadas pelo simulador. Esse número é a soma das iterações realizadas no processo de aprendizagem com as iterações realizadas no processo de inferência.

As funções membro da classe "Net" são:

```
Net(void);
~Net(void);
int Begin(void);
int MakeNode (int,int);
int MakeInput(int);
```

```
int MakeHidden(int);
int MakeOutput(int);
int MakeLink(int,int,int,int,float);
int DeleteUnit(int,int);
int DeleteLink(int,int,int,int);
int SetInput(int,float);
int SetOutput(int,float);
void Forward(void);
```

Sendo que:

Net(void) - inicializa as variáveis HiddenNumber, Inp, Out e Iteration com o valor 0.

~Net(void) - destrutor da classe Net que visa a liberação da memória anteriormente alocada para a rede.

Begin(void) - sua função principal é o de alocar memória para que a RNA possa ser criada pela ferramenta.
Em tendo sucesso, ela inicializa o campo Output de cada objeto Node da rede com o valor NONODE, indicando a não existência de EPs criados e retorna o valor 1.
Não tendo sucesso na alocação de memória, retorna o valor 0.

int MakeNode (int,int) - cria um EP através da inicialização das estruturas Output, Threshold e Delta com o valor 0 e os vetores Weights e LastWeigths com valor NOLINK. Estando coerente a determinação da posição do EP a ser criado dentro da rede, a função retorna o valor 1, estando incoerente retorna o valor -1.

int MakeInput(int) - cria a camada de entrada da rede através de várias chamadas à função MakeNode, atualizando a variável Inp com o valor 1 e Columm[0] com o número de EPs criados nesta camada. Se o número de EPs a serem criados nesta camada for menor ou igual a MAXCOLUMM, a função retorna o número de EPs criados, caso contrário retorna o valor -1.

int MakeHidden(int) - cria uma camada oculta de maneira análoga à função anterior utilizando a variável HiddenNumber. Retorna o número de EPs a serem criados se o mesmo for menor ou igual a MAXCOLUMM e se a rede ainda comportar a criação de mais uma camada oculta, caso contrário retorna o valor -1.

int MakeOutput(int) - cria a camada de saída da rede utilizando a variável Out. Atua de maneira análoga às duas funções anteriores, retornando o número de EPs criados nesta camada ou o valor -1 caso a especificação do número esteja incoerente.

int MakeLink(int,int,int,int,float) - cria uma ligação entre dois EPs desde que especificados os valores camada e posição dentro da camada, dos EPs inicial e final unidos pela ligação e o valor do peso da ligação. Retorna o valor um, caso as especificações sejam coerentes ou, o valor -1 caso contrário.

int DeleteUnit(int,int) - fornecida a posição de um EP da rede esta função o elimina da rede atribuindo-se à variável Output o valor NONODE. Elimina todas as ligações que entram e saem do EP a ser eliminado. Atualiza as variáveis Inp, Out e HiddenNumber e se for o caso também o vetor Columm. Retorna o valor 1 caso a localização especificada corresponda a um EP da rede.

int DeleteLink(int,int,int,int) - destrói a ligação unindo os dois EPs especificados, atribuindo à posição correspondente dos vetores Weights e LastWeights do EP inicial o valor NOLINK. Retorna o valor 1, caso a localização dos EPs esteja correta ou, o valor -1, caso contrário.

int SetInput(int,float) - atribui a uma posição do vetor Input um determinado valor correspondendo a um valor da camada de entrada da rede. Retorna o valor 1 caso a posição seja coerente com a camada de entrada.

int SetOutput(int,float) - análoga à anterior esta função atribui um valor à camada de saída da rede.

void Forward(void) - esta função realiza o mecanismo de inferência da rede, isto é, de posse dos valores da camada de entrada, esta função obtém os valores da camada de saída da rede.

VI.3.2 - Módulo Aprendizagem.

O Módulo Aprendizagem é o módulo responsável pelo processo de aprendizagem, ou seja, pelo ajuste dos valores dos pesos das ligações da rede.

Muito embora vários algoritmos de aprendizagem possam ser implementados no Módulo Aprendizagem, optou-se inicialmente pela implementação do algoritmo de "Backpropagation" por ser o mais representativo de uma ampla gama de algoritmos de aprendizagem.

Por apresentar atualmente somente um algoritmo de aprendizagem implementado, o Módulo Aprendizagem, diferentemente do Módulo Simulador, só possui uma classe, a classe Backpropagation.

VI.3.2.1 - A Classe Backpropagation.

A classe Backpropagation implementa como o próprio nome diz, o algoritmo de "Backpropagation" como descrito no artigo de Lippmann [Lip88].

Essa classe faz uso da rede implementada pela classe Net e de sua função membro Forward para realizar o processo de inferência o qual chamaremos da fase de "ida" do algoritmo.

Resta para a classe **Backpropagation** somente a tarefa de ajustar os pesos conforme o desempenho da rede. A esse processo de ajuste chamaremos de fase de "volta" do algoritmo.

Para realizar tal tarefa a classe apresenta as seguintes estruturas de dados e funções membros:

```
float Eta;
float Alfa;

Backpropagation(void) {Eta = Alfa = 0.5;}
void Backward (Net & ANN);
void Cycle (Net & ANN);
```

Onde:

Eta - representa o "termo de ganho" do algoritmo, utilizado no cálculo do ajuste dos pesos das ligações.

Alfa - representa o "termo momento" (*momentum term*) utilizado para tornar a convergência do algoritmo mais rápida.

Backpropagation(void) - construtor da classe que tem por objetivo a inicialização das variáveis Eta e Alfa com valor igual a (0,5). O valor da variável Alfa está limitado por definição ao intervalo 0 < Alfa < 1.

void Backward (Net & ANN) - verifica se a rede possui mais de uma camada e em caso positivo, realiza o processo de ajuste dos pesos da rede (fase de "volta").

VI.3.3 - Módulo de Armazenamento.

Este módulo é composto por uma única classe denominada **Save** que tem por objetivo implementar as operações de escrita e leitura das estruturas de dados da classe **Net** do Módulo Simulador em um arquivo externo.

Esta classe se presta, em última análise, à função de armazenar os processos de aprendizagem e/ou inferências realizados na rede criada no Módulo Simulador.

Como a manipulação de arquivos externos está sujeita a muitos erros tais como erros de leitura e escrita, a classe **Save** retorna em suas funções valores indicando o sucesso da operação ou o tipo de erro ocorrido.

A principal estrutura de dados desta classe é a variável **Arq** que armazena o nome do arquivo externo no qual serão armazenados os estados do Módulo Simulador durante os processos de aprendizagem e/ou inferência.

As principais funções membro desta classe são:

```
Save(char * file, Net & NN);
int File(short Status);
void SetBegin(void) {Position = COMECO; Offset = 0L;}
void SetCurrent(void){Position = CORRENTE; }
void SetEnd(void){Position = FIM; Offset = 0L;}
```

int Trunc(void);
int Read(Net & NN);
int Write(Net & NN);

Onde:

Save(char * file, Net & NN) - construtor da classe tem como função principal atribuir à variável **Arq** da classe Save, o nome do arquivo externo que será utilizado para armazenar as informações a ele fornecidas.

int File(short Status) - conforme o valor da variável Status, a função tenta criar e abrir o arquivo **Arq** , Status = CREATE, ou apenas abri-lo, Status = OPEN, para verificar a boa operacionalidade do mesmo em operações futuras, isto é, se é um arquivo existente e se permite operações de leitura e escrita.
Após a criação ou verificação do arquivo, a função termina por fechá-lo para a manutenção de sua integridade junto ao Windows.

void SetBegin(void) {Position = COMECO; Offset = 0L;} - ajusta o apontador do arquivo para a posição inicial do mesmo, isto é, quando ele for aberto, o será em seu início.

void SetCurrent(void){Position = CORRENTE;} - ajusta a posição do apontador do arquivo para a posição corrente do mesmo. Essa posição não necessariamente é o fim do arquivo, pois pode ter havido a operação de leitura do mesmo deixando o apontador apontado para o meio do mesmo.

void SetEnd(void){Position = FIM; Offset = 0L;} - ajusta o apontador do arquivo para a posição final do mesmo, esta situação ocorre quando se deseja gravar informações adicionais no mesmo.

int Trunc(void) - apaga o arquivo **Arq** a partir da posição indicada pelo apontador de arquivo.

int Read(Net & NN) - lê do arquivo **Arq** os vetores Input, Output e Columm e as variáveis HiddenNumber, Inp, Out e Iteration atribuindo seus valores às respectivas estruturas do Módulo Simulador. Armazena através da variável Offset a atual posição do arquivo.

int Write(Net & NN) - análoga à função **Read**, com a diferença que escreve os vetores e variáveis no arquivo ao invés de lê-los.

VI.4 - Bloco de Visualização.

O Bloco de Visualização tem como objetivo maior servir de interface entre o Módulo Simulador e o usuário, procurando possibilitar a este a manipulação dos recursos e a obtenção de informações de maneira clara e facilitada.

Passaremos a seguir a uma descrição de como esse bloco foi implementado no sentido de buscar atingir esses objetivos.

No Bloco de Visualização a cada janela implementada da ferramenta, existe associado pelo menos um módulo de programação que implementa todas as atividades representadas por esta janela.

Temos no Sirena doze módulos de programação correspondendo às doze janelas implementadas e mais três módulos de programação independentes (Entrada&Saída, DesenhodaRede e Seleção) que prestam auxílio aos outros módulos. (Veja figura VI.2)

Os módulos correspondentes às janelas possuem os nomes das mesmas, sendo, portanto, assim denominados:

- Janela Base
- Desenho da Rede
- Saída Estruturada
- Editor de Entrada
- Editor de Saída
- Saída dos EPs
- Limiares
- Pesos
- Pesos Anteriores
- Valores de Entrada da Rede
- Valores de Saída da Rede
- Situação da Rede

Descreveremos a seguir os módulos de programação pertencentes ao Bloco de Visualização começando com os módulos independentes e depois os associados diretamente às janelas.

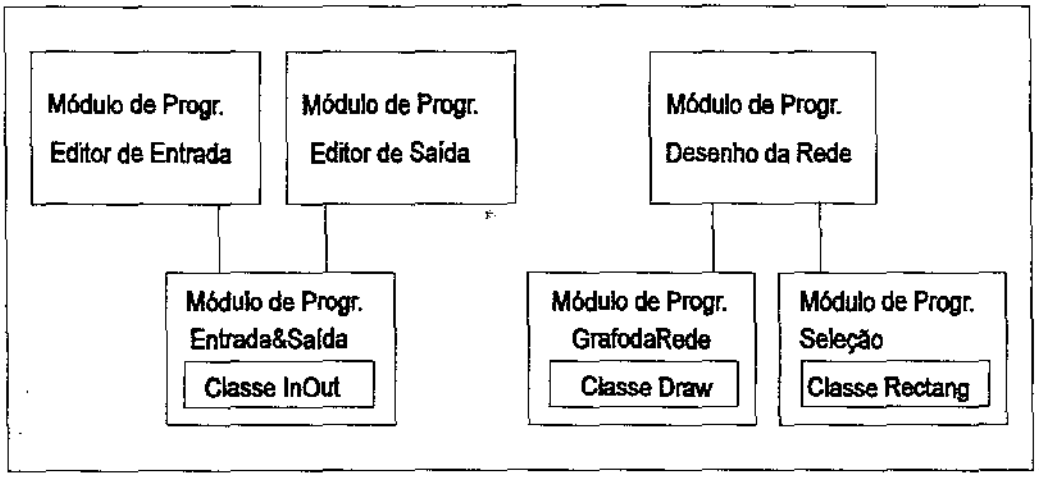


Figura VI.2 - Módulos Independentes e seus relacionamentos.

VI.4.1 - Módulo Entrada&Saída.

Este módulo é composto de uma única classe denominada **InOut** cujo objetivo é gerenciar os exemplos de entrada e saída desejados utilizados na aprendizagem da rede.

Para tal conta com as seguintes estruturas de dados:

```
short NumInExamp, NumOutExamp;
short InExample, OutExample;
float In [MAXEXAMPLES][MAXCOLUMM],
    Out [MAXEXAMPLES][MAXCOLUMM];
```

Onde:

NumInExamp e **NumOutExamp** - indicam respectivamente o número de exemplos de entrada e de saída já armazenados nas estruturas **In** e **Out**.

InExample e **OutExample** - indicam quais são os exemplos de entrada e saída correntes dentre os já armazenados.

In [MAXEXAMPLES][MAXCOLUMM],
Out [MAXEXAMPLES][MAXCOLUMM] - armazenam respectivamente todos os exemplos de entrada e saída desejados. O valor definido em MAXEXAMPLES corresponde ao limite máximo de pares de exemplos entrada/saída, sendo que na atual implementação este número corresponde a 10 pares.

As funções pertencentes a esta classe são:

```
InOut (void);
void StoreInput (int InOut, const Net & A);
void EraseInput (int InOut, const Net & A);
int NextInput (Net & A);
```

Onde:

InOut (void) - é o construtor da classe tendo como objetivo zerar as variáveis NumInExamp, NumOutExamp, InExample e OutExample.

void StoreInput (int InOut, const Net & A) - armazena um exemplo de entrada no vetor In caso InOUT = IN, ou de saída no vetor Out, caso InOut = OUT. Atualiza a variável NumInExamp ou NumOutExamp.

void EraseInput (int InOut, const Net & A) - o mesmo que StoreInput só que ao invés de armazenar, elimina o exemplo corrente e que está armazenado.

int NextInput (Net & A) - verifica se o número de exemplos de entrada é igual ao número de exemplos dos de saída. Se for, atribui aos vetores Input e Output do Módulo Simulador os próximos exemplos de entrada e de saída armazenados nas estruturas In e Out.

Atualiza as variáveis InExample e OutExample e retorna o valor 1.
Se o número de exemplos de entrada e de saída forem diferentes a função retorna o valor 0.

VI.4.2 - Módulo GrafodaRede.

Antes de passarmos à descrição deste módulo, mostraremos como o grafo que representa a rede é organizado na janela associada a este módulo.

Podemos ver na figura VI.3 que são utilizadas três entidades na apresentação da rede:

- *O Retângulo de Seleção* - esse retângulo representado na figura por ABCD é gerenciado pelo **Módulo Seleção** servindo, como explicaremos a seguir, para a seleção de um EP na rede.
  Cada EP, criado ou não no Módulo Simulador tem a ele associado um Retângulo de Seleção que determina na janela a posição onde o EP será desenhado.
  Todos os retângulos possuem o mesmo tamanho: largura SizeOfX e altura SizeOfY.
- *O Retângulo de Desenho* - está representado na figura por EFGH e tem como função limitar o tamanho do desenho do EP. Cada Retângulo de Desenho tem tamanho constante igual a SizeOfX/2 de largura e SizeOfY/2 de altura e está centrado no Retângulo de Seleção associado ao EP a ser desenhado. O desenho do EP está limitado, mas não determinado, pelo Retângulo do Desenho, isto é, ele pode ser igual ou menor que o Retângulo do Desenho.
- *A Distância Inicial* - determinada por Xinic e Yinic, indica qual é a distância que o primeiro Retângulo de Seleção, isto é, aquele que representa o primeiro EP da camada de entrada da rede, deve ter da moldura da janela.
  Essa distância, que na atual implementação é mantida constante, tem sua importância no cálculo de qual região da janela corresponde a um determinado Retângulo de Seleção.

O **Módulo GrafodaRede**, intimamente relacionado com o Módulo Desenho da Rede, tem a responsabilidade de desenhar o grafo da RNA criada.

Para tanto o módulo: cria todas as cores que podem ser utilizadas no grafo, desenha cada EP da rede com suas ligações na posição correspondente a ele na janela, com o formato desejado e com tamanho e cor variando ou não conforme seu valor de saída.

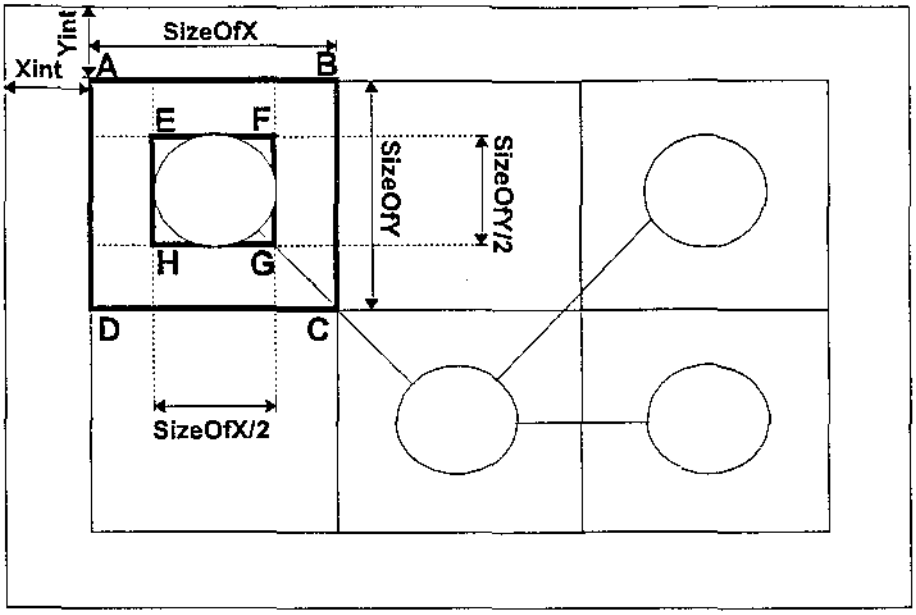


Figura VI.3 - Os retângulos de Seleção e Desenho na apresentação da RNA

O módulo é constituído pela classe **Draw** cujas estruturas de dados principais são:

```
LOGBRUSH lpBrush[100];
int   SizeOfX, SizeOfY;
int   Shape;
BOOL  Color;
BOOL  Size;
```

Onde:

lpBrush[100]- é o vetor que armazena as 100 possíveis cores a serem utilizadas na coloração dos EPs.

SizeOfX, SizeOfY - variáveis utilizadas no cálculo do Retângulo de Desenho conforme mostrado na figura VI.3.

Shape - variável que seleciona uma das duas formas (retângulo ou circunferência) implementadas de desenho do EP.

Color e Size - variáveis booleanas que indicam o desejo ou não da variação de cor e tamanho dos EPs conforme seus valores de saída.

Suas funções membro são:

```
void  CreateColors (void);
Draw (void);
void  DrawLink (HDC, NetPos, NetPos);
```

void DrawNode (HDC, NetPos, Net &);
void DrawNodeAndLink (HDC, int, int, Net &);
void DrawNet (HDC, Net &);

Onde:

CreateColors (void) - cria as cores utilizadas na pintura dos nós do grafo utilizando o padrão RGB de cores utilizado pela GDI (Interface de Dispositivos Gráficos) do Windows. Cada uma das 100 cores criadas é armazenada em uma posição do vetor lpBrush juntamente com o estilo (BS_SOLID) dos pincéis utilizados pelo Windows para a pintura dos EPs.

Draw (void) - construtor da classe inicializa as variáveis SizeOfX e SizeOfY com os valores padrões iguais a 100, a variável Shape com a forma "circunferência" e as variáveis Color e Size com o valor TRUE indicando o desejo de alteração de tamanho e cor dos EPs. Aciona a criação de cores através da chamada à função CreateColors.

DrawLink (HDC, NetPos, NetPos) - desenha a ligação entre os dois EPs dadas as suas posições dentro da rede.

DrawNode (HDC, NetPos, Net &) - para um EP determinado pelo parâmetro NetPos, a função realiza as seguintes atividades:

- determina o valor de saída do EP.
- obtém o pincel utilizado pelo Windows para a pintura do EP, selecionando o pincel preto se a variável booleana Color ajustada por meio da interface for igual a FALSE. Se Color = TRUE cria e seleciona o através do vetor lpBrush.
- determina a posição do Retângulo de Desenho.
- determina se a forma do EP a ser desenhado é circulo ou quadrado.
- determina o tamanho do EP que pode se manter fixo e com tamanho do Retângulo do Desenho se Size = FALSE ou variável com tamanho dependendo do valor de saída do EP, se Size = TRUE.
- desenha o EP e o colore com a cor selecionada.
- destrói o pincel criado.

DrawNodeAndLink (HDC, int, int, Net &) - dada a posição do EP dentro da rede, a função desenha, utilizando-se das duas funções anteriores, o EP desejado e todas as ligações que dele saem.

DrawNet (HDC, Net &) - para todos os EPs criados na rede, a função faz uma chamada a DrawNodeAndLink, desenhando desta forma a RNA completa.

VI.4.3 - Módulo Seleção.

Este módulo como o anterior também está vinculado ao Módulo Desenho da Rede. Tem como função a seleção de uma área da janela Desenho da Rede para a criação e/ou alteração do estado de um EP da rede, ou a seleção de duas áreas contíguas para a criação e/ou alteração do estado de uma ligação entre dois EPs em camadas sucessivas.

Para tanto o módulo faz uso da classe **Rectang** cujas principais estruturas de dados são:

```
struct NetPos
{
 int Row;
 int Columm;
} NPos;
NetPos FirstSelected;
NetPos LastSelected;
int   SelectedNumber;
```

Onde:

NPos - armazena a posição dentro da rede de um EP selecionado. É normalmente utilizada como uma variável auxiliar.

NetPos FirstSelected - armazena a posição dentro da rede do primeiro EP selecionado se dois EPs forem selecionados, ou a posição do único EP selecionado.

NetPos LastSelected - armazena a posição dentro da rede do segundo EP selecionado na janela caso sejam dois os EPs selecionados.

SelectedNumber - indica o número de EPs atualmente selecionados. Em qualquer instante pode não haver retângulo selecionado (SelectedNumber = 0), um retângulo selecionado para criação ou modificação de EP (Selected Number = 1) ou ainda, dois retângulos selecionados para a criação ou alteração de uma ligação (SelectedNumber = 2).

Suas funções membro são:

```
Rectang (void);
void Begin (HWND);
void XYtoNet (HDC, LONG);
void RectLimits (NetPos & , RECT &);
void FillRectan (HDC, RECT, int);
void FillSelected (HDC, int);
void RecttoNet  (RECT &, NetPos &);
BOOL IsNode (LONG, Net  &);
void Select (int, NetPos);
void SelectRectangle (Net  & );
```

void ClearAllSelections (void);
void ClearSelection (void);
void ChoseSelectClear (HDC, LONG, Net &);

Onde:

Rectang (void) - construtor da classe que tem como objetivo inicializar as variáveis FirstSelected e LastSelected com o valor -1 indicando que nenhum EP selecionado e SelectedNumber com o valor 0.

Begin (HWND) - informa à classe qual é o *"handle"*, utilizado pelo Windows, da janela Desenho da Rede.

XYtoNet (HDC, LONG) - dada a posição em que o *"mouse"* foi apertado, ato que indica o desejo de selecionar ou não mais selecionar um EP, a função:

- transforma essa posição de coordenadas físicas para coordenadas lógicas de tela, pois com a barra de rolagem da janela Visualização da Rede uma mesma posição física da tela pode corresponder a vários Retângulos de Seleção.
  Com a conversão para coordenadas lógicas esse problema desaparece pois cada coordenada lógica da tela corresponde a um único Retângulo de Seleção.
- transforma a coordenada lógica de tela na posição correspondente dentro da rede a qual o Retângulo de Seleção está associado.

RectLimits (NetPos & , RECT &) - dada a posição de um EP dentro da rede, a função devolve as coordenadas lógicas do Retângulo de Seleção associado.

FillRectan (HDC, RECT, int) - fornecida a posição lógica de um retângulo e a cor desejada, azul ou amarelo, a função cria o pincel correspondente e preenche o retângulo com a cor desejada.

FillSelected (HDC, int) - informado sobre a cor desejada preenche, usando a função *FillRectan*, os retângulos selecionados indicados por SelectedNumber e armazenados em FirstSelected e LastSelected.

RecttoNet (RECT &, NetPos &) - dadas as coordenadas lógicas do Retângulo de Seleção, devolve a posição dentro da rede a ele associado.

IsNode (LONG, Net &) - dada uma posição dentro da rede , verifica se nela existe ou não um EP criado.

Select (int, NetPos) - atualiza os campos das variáveis FirstSelected e LastSelected de acordo com a indicação FIRST (primeiro EP selecionado), ou LAST (segundo EP selecionado). Atualiza também a variável SelectedNumber.

SelectRectangle (Net &) - fornecida uma posição dentro da rede a função seleciona o Retângulo de Seleção correspondente se este retângulo possuir um EP criado na mesma

posição da rede, ou o primeiro Retângulo de Seleção disponível se não houver EP criado nesta posição.
Por primeiro disponível entenda-se a primeira posição disponível dentro de uma camada existente, seguinte ao último EP desta camada, ou a primeira posição da camada de saída seguinte à camada se, o desejo de seleção se der à direita desta camada.

ClearAllSelections (void) - Apaga todos os retângulos de seleção através da atribuição do valor -1 às variáveis FirstSelected e LastSelected indicando a não existência de retângulos selecionados e o valor 0 à variável SelectedNumber.

ClearSelection (void) - Apaga a seleção de um retângulo previamente selecionado. Se só houver um retângulo selecionado esta função faz uma chamada à função ClearAllSelections. Se houverem dois retângulos selecionados ela apaga a seleção do retângulo indicado em NPos e atribui ao outro retângulo a condição de primeiro retângulo selecionado.

ChoseSelectClear (HDC, LONG, Net &) - verifica se na posição onde o *"mouse"* é apertado existe um retângulo selecionado, em caso positivo a função apaga a seleção do retângulo através de ClearSelection, em caso negativo ela seleciona o retângulo através da função SelectRectangle.

Mostraremos a seguir os módulos de programação associados às janelas da interface Sirena.

VI.4.4 - Módulo Janela Base.

Todo o Bloco de Visualização da ferramenta Sirena está baseado na Interface de Múltiplos Documentos ou MDI (*Multiple Document Interface*) do Windows.

Na MDI existem três entidades: a **Janela de Base** ou Janela do Aplicativo Principal no nível mais alto, as **Janelas Filhas (JFs)** ou Janelas do Documento no nível mais baixo e a **Janela Cliente** gerenciando as outras e portanto, no nível intermediário. Como a Janela Cliente é gerenciada pelo Windows e fica invisível ao usuário, ela não merecerá mais comentários a seu respeito.

O **Módulo Janela Base**, como o próprio nome diz, implementa a Janela de Base da MDI, mas também realiza várias outras atividades tais como:

- Criação de todas as variáveis globais utilizadas pela ferramenta.
- Criação dos objetos das classes pertencentes ao Bloco de Simulação (Net, Backpropagation e Save) e ao Bloco de Visualização (Rectang, Draw e InOut).
- Tenta, através da função Begin da classe Net, alocar espaço para a RNA a ser criada. Não tendo sucesso, avisa o usuário e encerra o programa. Tendo sucesso, cria um EP na rede para que a mesma tenha pelo menos uma camada e um EP, evitando-se assim tentativas de aprendizagem e inferências em uma rede vazia.
- Criação e registro de classes de janelas no Windows para a Janela Base e para as Janelas Filhas.

- Cada Janela Filha é registrada a uma classe diferente, permitindo assim que cada uma possa ter atributos tais como ícone, *"menu"*, cursor e cor de fundo diferentes.
- Criação da Janela Base e a Janela Cliente fazendo com que a primeira apareça no monitor.
- Criação na Janela Base de um *"menu"* que será utilizado por todas as Janelas Filhas e que possibilita ao usuário:

  ⇒ A escolha das Janelas Filhas que lhe intessem em determinado momento da execução. Escolhida uma Janela Filha no *"menu"*, ela é automaticamente apresentada pelo Módulo Janela Base aparecendo no monitor sobre a Janela Base.

  ⇒ A organização das Janelas Filhas ou seus ícones na Janela Base (Cascata, Lado a Lado, Organizar Ícones).

  ⇒ A opção de encerramento da execução do programa com a respectiva destruição das Janelas Filhas.

  ⇒ A ativação de qualquer uma das Janelas Filhas criadas até o momento.
- Atualização de informações de todas as Janelas Filhas criadas através do mecanismo de atualização de informações ("REFRESH").
- Distribuição de mensagens recebidas para todas as janelas da ferramenta.
- Liberação após o encerramento, de todas as áreas de memória alocadas pela aplicação.

Os Módulos Janelas Filhas apresentam muitas semelhanças entre si, motivo pelo qual optamos por uma apresentação conjunta dos mesmos salientando suas características comuns e suas especificidades.

VI.4.5 - Características Comuns entre os Módulos Janelas Filhas (JFs).

Antes de descrevermos as especificidades dos módulos de programação das Janelas Filhas[1] descreveremos suas semelhanças e recursos compartilhados.

Por possuir uma estrutura diferente, algumas destas características comuns não se aplicarão à Janela Filha Desenho da Rede.

VI.4.5.1 - Criação.

No ato de criação de uma Janela Filha, basicamente duas atividades ocorrem:

[1] Trataremos daqui por diante os termos Módulos de Programação das Janelas Filhas e Janelas Filhas indistintamente.

- *A organização das JFs já existentes* - quando uma JF recém apresentada aparece no monitor, para que se mantenha uma organização na Janela Base, é enviada ao Windows uma mensagem para que ele disponha todas as JFs existentes lado-a-lado.
- *A ativação do mecanismo de atualização de informações (REFRESH)* - Toda JF tem como objetivo apresentar algum tipo de informação presente no Bloco da Simulação, por isso precisamos informar às JFs toda as vezes que as informações presentes neste bloco forem modificadas, isto é, o seu estado alterar.
  Cada JF apresenta o seu próprio mecanismo de atualização de informações, pois cada JF necessita de informações distintas do Bloco da Simulação.
  O mecanismo é ativado em duas situações distintas:
  ⇒ *Quando a JF é apresentada* - pois neste momento considera-se que todas as informações contidas em suas estruturas de dados estão desatualizadas.
  ⇒ *Quando uma JF altera o estado do Bloco da Simulação* - neste caso fica de responsabilidade da janela informar à Janela Base que ocorreu alguma alteração. A Janela Base de posse desta informação distribui mensagens a todas as JFs avisando-as para ativarem seus respectivos mecanismos de atualização de informações.

VI.4.5.2 - *"Menu"*.

Por pertencerem ao ambiente MDI, as JFs quando ativadas utilizam-se do "*menu*" da Janela Base. Tal "*menu*" apesar de poder ser alterado, como no caso da JF Desenho da Rede, foi mantido constante para as demais JFs do sistema.

O *"menu"* é composto de quatro itens:

- *Item Arquivo* - nele o usuário pode, da mesma forma que na Janela Base, criar cada uma das onze JFs disponíveis na ferramenta.
- *Item Registrar* - cria a caixa de diálogo Registro da Simulação, responsável pelo registro da aprendizagem/inferência da rede e também pelo retorno do simulador a uma iteração passada.
  Essa caixa de diálogo permite três atividades:
  ⇒ *Gravar a aprendizagem/inferência* - Através de duas variáveis booleanas, FileSave que demonstra o desejo ou não de se fazer a gravação e Recall, que demonstra se houve ou não o retorno a uma iteração passada gravada anteriormente.
  O critério sobre a gravação ou não da inferência/aprendizagem recai principalmente sobre a velocidade das mesmas, pois com a gravação a velocidade é menor.
  Fazendo-se uso do Módulo de Gravação, o mecanismo de gravação segue o seguinte esquema:
  Se existe o desejo de gravação (FileSave = TRUE) e houve retorno à iteração passada (Recall = TRUE) então o arquivo de gravação é truncado a partir daquela posição, pois assume-se que tudo o que está gravado anteriormente a essa iteração seja de

interesse e que a aprendizagem/inferência dali por diante tomará rumos diferentes, caso contrário não seriam regravadas as mesmas informações.
Se existe o desejo de gravação (FileSave = TRUE) mas não houve retorno a iteração passada (Recall = FALSE) então o arquivo de gravação é totalmente apagado para que se comece uma nova gravação a partir da iteração corrente.
Se não existe o desejo de gravação então só se alteram os estados das duas variáveis (FileSave e Recall) para FALSE.

⇒ *Mudança de nome do arquivo de gravação* - O nome do arquivo padrão de gravação está armazenado na variável DefaultFileName e o nome do arquivo externo está na variável Arq do Módulo de Armazenamento.
Desde que esta opção seja selecionada, pode-se criar um outro arquivo para a gravação da inferência/aprendizagem. Inicialmente a variável Arq possui o nome do arquivo padrão, havendo mudança de nome, a variável Arq passará a refletir a nova situação.

⇒ *Retorno a uma iteração passada* - Caso a iteração desejada seja uma iteração coerente, isto é, maior que zero, realiza-se uma busca desde o início do arquivo de gravação à procura da iteração desejada. Faz-se uso das mensagens de erro do Módulo de Gravação, caso ocorram erros nesta operação.
Termina por atualizar as variáveis Recall para TRUE indicando retorno a uma iteração anterior e FileSave para FALSE, o que força o truncamento do arquivo caso haja o desejo de uma nova gravação.
Uma mensagem é enviada à Janela Base confirmando mudança no estado do simulador e outra à Caixa de Diálogo Principal para o ajuste de seu visor.

- *Item Aprendizagem* - cria a Caixa de Diálogo Algoritmo de Aprendizagem, responsável pela alteração de parâmetros do algoritmo de "Backpropagation" e que permite a mudança das variáveis Alfa e Eta diretamente na classe Backpropagation do Módulo Aprendizagem.
- *Item Janela* - permite ao usuário a organização das JFs dentro da Janela Base (lado-a-lado, cascata, organizar ícones), fechar a janela ativa e ainda mudar o foco de entrada para uma das janelas já criadas através da escolha em um *"submenu"* contido neste.

VI.4.5.3 - Caixa de Diálogo Principal.

Toda Janela Filha, sem excessão, compartilha do recurso da Caixa de Diálogo Principal, que por ser "sem modo" permite que se mude de uma JF a outra sem que ela tenha que ser destruída.
Essa caixa possui quatro recursos:

- *Botão de Aprendizagem "Ciclo"* - realiza um ciclo do algoritmo de aprendizagem através das seguintes atividades:

⇒ Verifica através da função NextInput do Módulo Entrada&Saída, se o número de exemplos de entrada é o mesmo dos de saída.
Se não for avisa o usuário, e se for pega o próximo exemplo de entrada/saída.

⇒ Realiza um ciclo do algoritmo de "Backpropagation" fazendo uso da função Cycle do Módulo Aprendizagem.

⇒ Caso haja desejo de gravar a aprendizagem, isto é, FileSave = TRUE, grava-se usando a função Save do Módulo de Armazenamento. São tratadas todas as possíveis mensagens de erro.

⇒ Envia mensagem à Janela Base confirmando mudança do estado do simulador.

⇒ Atualiza o visor da própria Caixa de Diálogo Principal.

- *Botão de Aprendizagem "Vários"* - realiza vários ciclos do algoritmo de "Backpropagation" através das mesmas atividades do Botão "Ciclo" sendo que o envio da mensagem e atualização do visor só ocorrem após todos os ciclos terem sido executados.
- *Botão "Inferência"* - realiza uma inferência fazendo-se uso da função Forward do Módulo Simulador. Verifica o desejo de gravação da inferência através da variável FileSave e ao final, atualiza o visor da caixa de diálogo e envia mensagem de mudança de estado à Janela Base.
- *Visor de Iterações* - mostra o número de iterações feitas pelo Módulo Simulador através de sua variável Iteration descrita no Módulo Simulador. Caso haja um retrocesso a uma iteração passada, como o estado do simulador refletirá está iteração, o Visor de Iterações também será alterado.

VI.4.6 - Especificidades dos Módulos Janelas Filhas.

As Janelas Filhas da ferramenta se dividem em dois grupos: *as janelas textuais* (Saída dos EPs, Limiares, Pesos, Pesos Anteriores, Valores de Entrada da Rede, Valores de Saída da Rede e Situação da Rede) e *as janelas gráficas* (Visualização da Rede, Saída Estruturada, Editor de Entrada e Editor de Saída).

As JFs textuais apresentam estrutura muito semelhante, sempre obtendo informações do Bloco de Simulação e as apresentando no monitor de maneira escrita, não merecendo, portanto, comentários mais aprofundados sobre seu funcionamento.

As três Janelas Filhas gráficas: Editor de Entrada, Editor de Saída e Saída Estruturada apresentam estruturas semelhantes podendo dessa forma serem discutidas conjuntamente, enquanto que, a JF Desenho da Rede exige comentários à parte.

VI.4.6.1 - Janelas Filhas Editor de Entrada, Editor de Saída e Saída Estruturada.

As três JFs têm como estrutura de dados principal a matriz

static short fState [xDIVISIONS][yDIVISIONS]

que quadricula a janela em questão em xDIVISIONS * yDIVISIONS retângulos. Este número que corresponde ao máximo de EPs permitidos tanto na camada de entrada quanto na camada de saída da rede.

Na atual implementação esse número que é definido em MAXCOLUMM, corresponde a 15, resultando em uma divisão de xDIVISIONS = 3 e yDIVISIONS = 5.

Cada retângulo, que é associado a um elemento da matriz **fState**, corresponde:

- a um elemento do vetor Input no Módulo Simulador se for a JF Editor de Entrada,
- a um elemento do vetor Output, se for a JF Editor de Saída
- ou ainda a um EP da camada de saída da rede se for a JF Saída Estruturada.

Cada retângulo pode assumir quatro estados:

COR	Valor de fSate[x][y]	Significado
cinza claro	-1	Não existe EP criado nesta posição
branca	0	0 <= valor do EP <= 0,1
cinza escuro	1	0,1 < valor do EP < 0,9
preta	2	0,9 <= valor do EP <= 1

A JF Saída Estruturada apresenta a característica de acionar um aviso sonoro caso o usuário aperte o *"mouse"* em um dos seus retângulos. Esta característica diferencia a JF Saída Estruturada das outras duas JFs.

As JFs Editor de Entrada e Editor de Saída se caracterizam pela mudança cíclica de cor de retângulo toda vez que um retângulo diferente de cinza escuro é selecionado.

A alteração de cor é acompanhada de alteração de valor de fState que assume ciclicamente os valores 0, (0,5) e 1.

Acertados os padrões de entrada na JF Editor de Entrada e de saída na JF Editor de Saída um toque com o botão direito do *"mouse"* em qualquer das duas JFs, ativará a caixa de diálogo que permite a gravação ou eliminação de um exemplo de entrada/saída. Tal caixa faz uso das funções do Módulo Entrada&Saída.

VI.4.6.2 - Janela Filha Desenho da Rede.

A JF Desenho da Rede objetiva mostrar a topologia e o estado da RNA através do grafo que a representa, sendo que o desenho deste grafo é obtido através de uma chamada à função DrawNet do Módulo GrafodaRede.

Dentre as JFs esta destaca-se por apresentar o *"submenu"* "Ajustar" responsável pela criação da Caixa de Diálogo Ajustar.

Esta caixa de diálogo destina-se ao controle de como os Módulos de Programação Seleção e DesenhodaRede apresentam seus resultados na JF Visualização da Rede.

Os resultados podem variar em três categorias:

- *tamanho do Retângulo de Seleção com a conseqüente variação do tamanho dos EPs* - onde conforme a escolha (pequeno, médio pequeno, médio grande e grande), as variáveis SizeOfX e SizeOfY do módulo DesenhodaRede e, SizeX e SizeY do módulo Seleção recebem os valores: 25, 50, 100 e 150.
- *forma dos EPs apresentados* - onde dependendo da opção (circunferência e quadrado) a variável Shape do Módulo GrafodaRede recebe os valores CIRCLE ou SQUARE.
- *a variação ou não de tamanho e/ou cores dos EPs conforme os valores de saída dos mesmos* - onde as variáveis Size e Color do Módulo GrafodaRede recebem o valor TRUE, indicando desejo de variação de tamanho ou o FALSE, indicando o desejo de constância de tamanho.

Apertando-se o botão esquerdo do *"mouse"* em qualquer parte da área do cliente da JF Desenho da Rede, um Retângulo de Seleção é ativado ou apagado de acordo com a função ChoseSelectClear do Módulo de Programação Seleção.

Apertando-se o botão direito e tendo-se um retângulo selecionado é apresentada a caixa de diálogo "Cria e Altera EP" cujo objetivo é criar e/ou alterar o estado do EP cuja posição corresponda ao retângulo selecionado.

Esta caixa primeiramente verifica se existe interesse por parte do usuário em deletar o EP selecionado, se houver ela o faz por meio de uma chamada à função DeleteUnit do Módulo Simulador.

Se não houver interesse e se a posição selecionada não contiver um EP, ela o criará por meio da função MakeNode do Módulo Simulador.

Apertando-se com o botão direito do *"mouse"* e tendo-se dois retângulos selecionados na JF Desenho da Rede, é apresentada a caixa de diálogo "Cria e Altera Ligação" que tem como objetivo permitir a criação e/ou alteração do valor do peso da ligação que une os dois EPs correspondentes aos retângulos selecionados.

Esta caixa de diálogo verifica o desejo de eliminação de uma ligação. Caso ele exista, a caixa o executará através de uma chamada à função DeleteLink do Módulo Simulador, em caso contrário uma nova ligação será criada através da função MakeLink do MóduloSimulador.

Os valores das janelas de edição se tornarão os novos valores de peso e peso anterior da ligação entre os dois EPs selecionados.

Em qualquer dos casos, todo retângulo selecionado é apagado com a chamada à função ClearAllSelections do Módulo de Programação Seleção.

Sempre que o estado do simulador é alterado pelas caixas de diálogo da JF Desenho da Rede, a Janela Base será informada desta alteração para que a mesma possa ativar os mecanismos de atualização de informações das demais janelas.

VI.5 - CONSIDERAÇÕES FINAIS.

Antes de encerrarmos o capítulo faremos alguns comentários, a nível macroscópico, a respeito da atual implementação.

O Bloco da Simulação foi implementado de maneira robusta e simples onde um módulo simula, outro aprende e o outro armazena e recupera informações, sendo que por terem sido assim criados, eles podem ser facilmente modificados e expandidos.

O Módulo Simulador, responsável pela criação e manutenção da rede, possui uma estrutura de dados bem definida onde alterações nos EPs são feitas na classe Node e alterações na estrutura da rede são feitas na classe Net.

Suas funções são conceitualmente simples (criam, destroem ou ajustam alguma entidade) e portanto, podem ser facilmente alteradas. Em uma implementação mais complexa elas poderiam ser declaradas como virtuais para que em uma classe derivada estas funções pudessem até mesmo ser substituídas por outras.

Neste módulo alterações a serem pensadas são: a inclusão da possibilidade de escolha das funções de ativação e transferência por parte do usuário no momento de criação da rede e a implementação da mesma como um grafo e não como uma matriz, permitindo assim a criação de redes mais genéricas e com menor alocação de memória ao custo, porém, de uma velocidade de execução menor.

Quanto à questão do tamanho máximo da rede, como todos os módulos do Bloco da Simulação estão vinculados ao gerenciamento de memória feita pelo Windows, alocações de memória precisam ser requisitadas ao mesmo, sempre correndo o risco de se ter o pedido rejeitado por algum motivo alheio ao simulador.

O Módulo Aprendizagem atualmente constituído por apenas um algoritmo, utiliza a estrutura criada pelo Módulo Simulador, a ele, portanto, resta apenas o papel da aprendizagem da rede.

A introdução de novos algoritmos é simples onde cada novo algoritmo corresponde a uma nova classe do módulo. Estas classes têm plena liberdade de manipulação da rede, já que esta é toda pública, isto é, permite acesso irrestrito a suas estruturas.

Quanto ao Módulo de Armazenamento, o único comentário a ser feito é que alterações (inclusão e eliminação) de estruturas da rede devem ser acompanhadas de alterações nas funções Read e Write deste módulo, pois funções lêem e escrevem as estruturas da rede em um arquivo externo.

Vale aqui também a sugestão de tornar tais funções virtuais em um a implementação mais complexa.

Os maiores comentários a respeito do Bloco da Visualização são aqueles envolvendo critérios de funcionalidade, feitos, portanto, na parte final do Capítulo V sobre a descrição dos recursos do Sirena e sua utilização.

Embora funcionalmente sem problemas, a implementação deste bloco pode ser robustecida com a criação de objetos gráficos do Windows ou com a utilização dos objetos fornecidos pelos modernos compiladores .

Tais objetos que representam recursos gráficos tais como janelas, botões e caixas de diálogo, apresentam a característica de facilitar em muito a criação, manipulação e destruição de tais recursos, pois realizam, sem a necessidade de intervenção por parte do programador, tarefas de mais baixo nível tais como registro de classes de janelas, gerenciamento de mensagens e tratamento de exceções.

Com esse tipo de objetos, as alterações da interface gráfica do Sirena poderá ser feita em um nível mais alto do que o permitido pela atual implementação, permitindo ao programador concentrar-se nos aspectos funcionais da interface.

De qualquer forma, toda alteração da interface atual incorre em algum dos dois casos: a modificação dos recursos existentes e a inclusão de novos recursos.

No primeiro caso basta que o programador modifique o recurso gráfico e o respectivo código de programação que o gerencia.

Já no segundo caso além da criação do recurso gráfico e seu respectivo código, o programador também fica responsável pela criação do mecanismo de atualização de informações deste recurso.

Se o novo recurso for uma janela, resta o trabalho adicional de incluí-la no *"menu"* da Janela Base para que a mesma possa ser ativada pelo usuário em algum momento de utilização da ferramenta.

Com relação ao código como um todo, isto é, Bloco de Simulação e de Visualização, apresenta uma velocidade apropriada quando o mecanismo de armazenamento de informações não está acionado e quando a capacidade de processamento da máquina não é muito baixa com o Windows.

Finalmente quanto à portabilidade da ferramenta, todas as máquinas com capacidade de executarem a versão 3.1 do Windows com desenvoltura poderão executar a ferramenta sem dificuldades.

Capítulo VII

Conclusões

Este capítulo apresenta as conclusões finais deste trabalho assim como sugestões para extensões futuras da versão atual do Sirena.

VII.1 - Conclusões Gerais.

Conforme enfatizado ao longo deste trabalho, a maior dificuldade no entendimento e manipulação das RNAs está no desconhecimento de como as mesmas armazenam informações em suas estruturas e de como utilizam essas informações para a realização do processo de inferência.

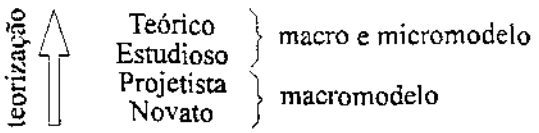
Embora matematicamente tenhamos uma idéia geral sobre seu funcionamento, ela não é forte o suficiente para que possamos desenvolver critérios científicos que nos determinem qual a melhor topologia da rede, qual o melhor algoritmo de aprendizagem e qual a melhor amostragem de informações a serem fornecidas para uma determinada aplicação.

O desenvolvimento de novos modelos de RNAs e de novos algoritmos de aprendizagem fica prejudicado com esse não entendimento completo do funcionamento das RNAs, pois qualquer que seja o processo de melhoramento parte do pressuposto que saibamos *"a priori"* qual é a imperfeição a ser melhorada.

Caso continuem atuando como uma "caixa preta", onde ainda concluímos critérios de eficiência e capacidade de processamento baseados quase que totalmente na observação das informações apresentadas na camada de saída, as RNAs dificilmente chegarão à condição de ferramentas de uso geral com fácil acesso a todos, como o que ocorre com os computadores convencionais.

Enquanto uma metodologia geral de entendimento do funcionamento das RNAs não for totalmente desenvolvida, supondo-se ser esta uma tarefa viável, devemos delimitar o universo dos usuários das redes de forma a aplicar soluções particulares para cada um dos grupos de usuários.

Conforme nossa proposta, delimitamos os usuários em quatro grupos e os ordenamos em ordem crescente de domínio teórico a respeito do assunto: o novato no campo das RNAs, o projetista, o estudioso e finalmente o teórico.



Acreditamos que quaisquer que sejam as soluções propostas, devam elas de alguma maneira promover um modelo mental [GS83] no usuário a respeito do funcionamento das redes, sendo que tal modelo mental provavelmente estará muito relacionado com o nível de teorização do usuário.

Enquanto que para o novato na área e para o projetista de uma rede, um entendimento geral do funcionamento das RNAs, o qual chamaremos de *macromodelo*, deva ser suficiente, para o estudioso e para o teórico, porém, além do entendimento geral, um entendimento mais minucioso do mesmo, o *micromodelo*, será necessário.

O nosso trabalho visa contribuir para a criação destes modelos, o macro e o micromodelo, seja através do fornecimento efetivo de ferramentas que auxiliem nesta tarefa ou pela indicação dos aspectos mais complicados envolvendo a criação de tais ferramentas, objetivando incentivar o estudo dos mesmos para uma futura busca de soluções.

Para o fornecimento destas ferramentas o Sirena, dentre outros possíveis meios, utilizou-se do ambiente computacional, pois o mesmo pode ser desenvolvido de forma a oferecer facilidades para o entendimento das RNAs. Dentre estas facilidades podemos destacar:

- *Dinâmica de processo* - onde as informações sobre as RNAs são apresentadas de maneira não estática, muitas vezes no momento em que são geradas, favorecendo com isso o entendimento dos processos envolvidos (aprendizagem/inferência) como uma seqüência de subprocessos.
- *Facilidade de incrementar recursos* - uma vez detectada a necessidade de modificação ou acréscimo de recursos existentes, o ambiente computacional fornece ferramentas que de maneira rápida e facilitada permitem que as alterações no sistema sejam feitas de maneira harmônica, isto é, os recursos modificados e/ou acrescidos acabam por não prejudicar a funcionalidade dos outros recursos pré-existentes e nem o *"layout"* da interface.
- *Facilidade e rapidez de avaliação dos recursos implementados* - por apresentar uma alta interação com o usuário, pode-se através da observação das reações do mesmo avaliar a eficiência dos recursos ao mesmo tempo que se obtém indicações para a melhoria de tais recursos.
- *Atratividade das ferramentas* - talvez nenhum outro meio que não o computacional, tenha condições de oferecer tantos recursos (imagem com qualidade, animação, som, interatividade) que motivem o usuário a lidar com todos os conceitos envolvidos no processo de entendimento das RNAs.
- *Coexistência de ambientes* - o mesmo ambiente utilizado pelo usuário para o entendimento dos conceitos das RNAs é também o utilizado para o desenvolvimento de aplicações envolvendo as mesmas. O usuário pode então utilizar-se da filosofia do "aprender trabalhando" pois o ambiente é projetado de forma a facilitar que estes processos possam ocorrer simultaneamente.

No Sirena as ferramentas voltadas para a formação do macromodelo são basicamente as ferramentas gráficas pois estas foram desenvolvidas de forma a fornecer, na medida do possível, informações qualitativas a respeito da RNA simulada.

A utilização de representações qualitativas foi embasada no conhecimento que se tem sobre a importância do raciocínio qualitativo em se tratando de novatos em um dado domínio [Bob84][For93][Kui93][Ste92].

Para a formação do micromodelo, destinam-se principalmente as ferramentas textuais e de armazenamento de informações, pois estas visam fornecer informações quantitativas e portanto, precisas sobre a rede.

Na formação do macro e micromodelo devem ser utilizados os dois tipos de ferramenta para que se tenha uma visão geral (qualitativa) e uma visão pormenorizada (quantitativa) ao mesmo tempo. A oportunidade de visualizar representações alternativas e complementares é essencial ao entendimento de qualquer fenômeno simulado.

Verificamos no desenvolvimento do Sirena que as ferramentas voltadas para o entendimento do macromodelo, embora exigindo tanta criatividade quanto se deseje, são de mais fácil construção uma vez que se supõe serem destinadas a usuários sem maiores bases teóricas, limitando, portanto, o conjunto de elementos (conceitos) a serem manipulados pela ferramenta.

O micromodelo por sua vez exige informações detalhadas e precisas, voltadas para o usuário com uma grande base teórica, com uma postura mais crítica e uma necessidade de cruzamento maior de informações.

As ferramentas voltadas para este modelo, portanto, exigem uma complexidade muito maior na medida que tentam transmitir conceitos, muitas vezes envolvendo elementos multidimensionais, de maneira facilitada sem que haja nenhuma perda no conteúdo da informação.

Salientamos que as ferramentas sozinhas não constituem material suficiente para a criação de modelos de RNAs em pessoas sem nenhum conhecimento no assunto, pois o Sirena foi pensado de forma a dar liberdade ao usuário em lidar, sem receios, com um conjunto inicial de conceitos, seja ele pequeno ou não, e a partir dele formar novos conceitos para ampliar o conjunto inicial.

Enquanto protótipo o Sirena apresenta características cuja presença consideramos ser importante em qualquer projeto cujos objetivos assemelham-se aos nossos. Tais características são:

- *Interação com o usuário* - onde o mesmo pode modificar de maneira facilitada a topologia da rede e os parâmetros do algoritmo de aprendizagem, retornar a iterações passadas e ajustar a forma de apresentação das informações conforme o seu desejo sempre recebendo do sistema informações sobre as conseqüências das modificações.
- *Cruzamento de informações* - seja através de várias janelas ativas da ferramenta, seja através do mecanismo de armazenamento de informações, o usuário cruzando as informações e representações apresentadas tem condições de tirar suas próprias conclusões (é quando dizemos que ocorreu uma aprendizagem por parte do mesmo).
- *Liberdade de manipulação dos recursos* - o usuário tem acesso de maneira facilitada a todos os recursos da ferramenta, podendo utilizá-los na quantidade, ordem e momento

que mais lhe convier, permitindo ao mesmo o controle total sobre o funcionamento da mesma.

- *Visão qualitativa* - a possibilidade de entender os processos (aprendizagem, inferência, disparo) e as entidades (EPs, RNA) sem a necessidade de utilização do conceito de quantidade (recursos numéricos), permite ao usuário perceber a mudança de estado da rede e a evolução das mudanças sem necessitar precisar com exatidão qual é realmente o estado atual da mesma.

Finalizamos em dizer que o Sirena, ao invés de uma solução, é um conjunto de idéias no sentido de subsidiar um passo inicial no caminho do entendimento das RNAs, e com tal deve passar por fases de evolução, de forma a poder cada vez mais atingir os seus objetivos.

Indicaremos a seguir algumas sugestões para uma primeira etapa de melhoramentos.

VII.2 - Extensões e Trabalhos Futuros.

A avaliação em campo dos recursos oferecidos pelo Sirena, embora tendo sido testados em menor escala, é a continuação natural deste trabalho, pois somente a partir da determinação precisa das deficiências atuais é que poderemos realizar alterações e extensões eficientes.

De nossa experiência atual pudemos constatar que várias são as possíveis extensões/modificações à implementação atual do Sirena, algumas delas visando aumentar a facilidade de uso e outras aumentar o seu poder de simulação. Devemos, porém, ter sempre em mente o compromisso entre facilidade de entendimento e uso da ferramenta e a quantidade de recursos oferecidas pela mesma, pois a facilidade de entendimento do funcionamento global dos recursos presentes na ferramenta diminui proporcionalmente ao aumento do número de facilidades oferecidas.

A seguir citamos algumas delas, tomando o cuidado de dividi-las em dois grupos conforme o objetivo a que se destinam: a formação do macro ou a do micromodelo.

Dentre as extensões/modificações orientadas para o macromodelo podemos sugerir:

- *Aprimoramento da atual interface gráfica* - tornando a interface cada vez mais transparente ao usuário [Ris87], favorecendo ao mesmo concentrar-se somente na formação do seu modelo mental da rede. Como ponto de partida poderiam ser utilizados alguns problemas encontrados na atual interface, tais como a associação de cor/intervalo e a velocidade de acesso a recursos com a utilização de um botão adicional do *"mouse"* (para uma discussão complementar sobre estes tópicos veja considerações finais dos capítulos V e VI). Toda esta análise deve ser feita "à luz" de novas idéias sobre "interfaces centradas no usuário" e/ou "interfaces centradas na tarefa", como apresentadas em [Lau93].
- *Janelas Editor de Entrada, Editor de Saída e Saída Estruturada apresentando somente EPs criados* - ao invés de mostrar o número máximo de EPs das camadas de entrada e

saída, estas janelas apresentariam somente o número atual de EPs, favorecendo assim o processo de introdução de informações e de visualização do formato geométrico das mesmas.

- *Diferenciação da Caixa de Diálogo Principal para cada janela* - permitiria ao usuário um acesso fácil e diferenciado aos recursos específicos em cada janela, facilitando ainda mais a utilização da ferramenta.
- *Help contextual* - criação de uma ajuda explicativa acionada pelo usuário posicionando o *"mouse"* sobre o recurso a ser esclarecido, facilitando com isso o entendimento do funcionamento dos recursos da ferramenta.
- *Help teórico* - um auxílio destinado aos novatos que mostraria através de vários recursos como animação e "multimídia", algumas informações básicas a respeito das RNAs para motivá-lo no uso dos demais recursos.
- *Visualização do estímulo total de entrada dos EPs feito através da variação de cor e/ou tamanho dos nós do grafo* - permitiria um melhor entendimento do funcionamento da entidade limiar dos EPs.
- *Visualização de EPs não alterados* - destacaria no grafo em cada iteração do algoritmo de aprendizagem ou em cada inferência, os EPs que não dispararam, objetivando com isto mostrar áreas menos facilitadas da rede com o processamento de determinadas informações.
- *Variação do formato do desenho dos nós conforme os valores de saída dos EPs* - opção adicional de visualização dos valores de saída dos EPs que poderia ser acompanhada das já existentes (cor e tamanho) de forma a facilitar ainda mais a visualização das variações de valores dos EPs.
- *Visualização da variação dos valores dos pesos através da coloração das ligações* - em uma janela adicional onde os nós estariam constantes, as ligações variariam de cor obedecendo certas faixas de valores para os pesos. Este mecanismo mostraria qualitativamente a informação sendo armazenada na rede.

Dentre as extensões para a formação do micromodelo sugerimos:

- *Separação na interface dos recursos de acordo com níveis de sofisticação* - permitiria que recursos cada vez mais sofisticados pudessem ser adicionados à ferramenta sem que os mesmos complicassem a utilização da ferramenta nas situações onde esses recursos não seriam utilizados, isto é, em níveis de entendimento mais baixos.
- *Armazenamento das informações de entrada e saída em local diferente das informações da simulação* - permitiria mais facilmente que os processos de simulação e aprendizagem fossem realizados com várias informações de entrada diferentes a partir de uma mesma topologia da rede.
- *Simulação comparada* - onde mais de um modelo de rede pudessem ser simulados e alterados ao mesmo tempo possibilitando a comparação dos resultados das redes à medida que os resultados fossem sendo produzidos.

- *Extensão da rede simulada para redes mais gerais onde cada EP pudesse se ligar com qualquer outro EP da rede* - permitiria a construção de topologias cada vez mais gerais possibilitando entendimentos mais complexos do funcionamento das RNAs.
- *Extensão para mais algoritmos de aprendizagem* - permitindo ao usuário um conhecimento das diferenças de funcionamento dos diversos algoritmos de aprendizagem.
- *Alteração em tempo de execução das funções de ativação e transferência para cada EP* - permitiria ao usuário um conhecimento do comportamento das funções, além de um controle maior da rede através da diferenciação de funcionamento dos EPs.
- *Visualização das funções de ativação e transferência através de gráficos em cada EP* - permitiria um melhor entendimento do funcionamento das mesmas e de sua adequação a uma determinada rede.

Concluindo, acreditamos fortemente, que esforços devem ser despendidos no sentido de tornar mais claro o funcionamento de RNAs. A obscuridade de seu processamento no sentido de como é representado o conhecimento adquirido durante o processo de aprendizagem, tem sido sem dúvida a principal barreira no avanço das aplicações e fundamentalmente na interligação de aplicações típicas RNA com aplicações típicas simbólicas.

Referências Bibliográficas

[Avi87] ÁVILA, G. - Funções de Várias Variáveis. In: _____ - **Cálculo - Funções de Várias Variáveis.** 4.ed. Livros Técnicos e Científicos Editora S. A., 1987. p.45-84.

[Bob84] BOBROW, D.G. - Qualitative Reasoning about Physical Systems: an Introduction. **Artificial Intelligence., 24(1-3):**1-5, 1984.

[CM85] CHARNIAK, E. & MCDERMOTT, D. - Learning. In: _____ - **Introduction to Artificial Intelligence.** Addison-Wesley Publishing Company, 1985. p.609-662.

[CS88] COWAN, J.D. & SHARP, D.H. - Neural Nets and Artificial Intelligence. **Daedalus., 117:**Winter, p.85-121, 1988.

[Cun85] CUN, Y.L. - A learning scheme for asymmetric threshold networks. In: **Proceedings Cognitiva 85.** Paris, France, 1985 p.599-604.

[Dav87] DAVIS, K., et al., eds, **Longman dictionary of contemporary English.** 2.ed., Longman Group UK Limited, 1987. 1129p.

[Die90] DIEDERICH, J. Artificial Neural Networks: Concept Learning. An Introduction. In: DIEDERICH, J., ed. - **Artificial Neural Networks: Concept Learning.** IEEE Computer Society Press, 1990. p.1-10.

[Fir88] FIREBAUGH, M.W. - New Architectures for AI. In: _____ - **Artificial Intelligence- A Knowledge-Based Approach.** Boyd & Fraser Publishing Company, 1988, p.634-672.

[For93] FORBUS, K.D. - Qualitative process theory: twelve years after. **Artificial Intelligence., 59:**115-123, 1993.

[GS83] GENTNER, D. & STEVENS, A.L., ed. - **Mental Models.** London, Lawrence Erlbaum Assoc. Publish, 1983.

[Guy76] GUYTON, A. C. - **Fisiologia Humana.** Interamericana, 1976.

[Heb49] HEBB, D.O. - **The Organization of Behavior.** New York, John Wiley, 1949.

[Hin90] HINTON, G.E. Connectionist Learning Procedures. In: DIEDERICH, J., ed. - **Artificial Neural Networks: Concept Learning.** IEEE Computer Society Press, 1990. p.11-47.

[HN88] HECHT-NIELSEN, R. - Neurocomputing: picking the human brain. In: VEMURI, V. ed. - **Artificial Neural Networks: Theoretical Concepts.** Computer Society Press of IEEE, 1988. p.13-18.

[Kap82] KAPLAN, W. - Cálculo Diferencial de Funções de Várias Variáveis. In: _____ - **Cálculo Avançado vol I.** 2.ed. Editora Edgard Blücher Ltda, 1982. p.82-153.

[Ken90] KENTRIDGE, R.W. - Neural networks for learning in the real world: representation, reinforcement and dynamics. **Parallel Computing., 14**:405-414, 1990.

[Koh88] KOHONEN, T. - An Introduction to Neural Computing. **Neural Networks. 1**:3-16, 1988.

[Kui93] KÜIPERS, B. - Reasoning with qualitative models. **Artificial Intelligence, 59**:125-132, 1993.

[Lau93] LAUREL, B., ed. - **The Art of Human-Computer Interface Design.** Addison-Wesley Publish Co, 1993.

[Lev89] LEVINE, D.S. - The Third Wave in Neural Networks. **AI Expert., december**:27-31, 1989.

[Lip88] LIPPMANN, R.P. - An Introduction to Computing with Neural Nets. In: VEMURI, V., ed. - **Artificial Neural Networks: Theorical Concepts.**, Computer Society Press of IEEE, 1988. p.36-54.

[MP43] MCCULLOCH, W.S. & PITTS, W.H. - A Logical Calculus of the Ideas Immanent in Nervous Activity. **Bulletin of Mathematical Biophysics., 5**:115-133, 1943.

[Par85] PARKER, D.B. - Learning-logic. **Tech. Rept TR-47**, Sloan School of Management, MIT, Cambridge, MA, 1985.

[Pet93] PETZOLD, C. - A Interface de Múltiplos Documentos (MDI) In: _____ - **Programando para Windows 3.1.** MAKRON Books do Brasil Editora Ltda, 1993. p.935-959.

[PM69] PAPERT, S. & MINSKY, M. - **Perceptrons: An Introduction to Computational Geometry.** Cambridge: MIT Press, 1969.

[RC90] REILLY, D.L. & COOPER, L.N. - An Overview of Neural Networks: Early Models to Real World Systems. In: ZORNETZER, S.F.; DAVIS, J.L.; LAU, C., eds - **An Introduction to Neural and Eletronic Networks.** Academic Press, Inc., 1990. p.227-248.

[RHM86] RUMELHART, D.E.; HINTON, G. E.; McCLELLAND, J.L. - The Appeal of Parallel Distributed Processing. In: RUMELHART, D.E. & McCLELLAND, J.L., ed. - **Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.** 5.printing. MIT Press, 1986. p.3-44.

[RHM86a]RUMELHART, D.E.; HINTON, G. E.; McCLELLAND, J.L. - A General Framework for Parallel Distributed Processing. In: RUMELHART, D.E. & McCLELLAND, J.L., ed.- **Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.** 5.printing. MIT Press, 1986. p.45-76.

[RHW86] RUMELHART, D.E.; HINTON, G. E.; WILLIAMS, R.J. - Learning Internal Representations by Back-propagating Errors., **Nature., 323:**533-536, 1986.

[RHW86a]RUMELHART, D.E.; HINTON, G. E.; WILLIAMS, R.J. - Learning Internal Representations by Error Propagation. In: RUMELHART, D.E. & McCLELLAND, J.L. ed. - **Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.** 5.printing. MIT Press, 1986. p.318-364.

[Ris87] RISSLAND, E.L. - Ingredients of Intelligent User Interface. In: BAECKER, R.M. & BUXTON, W.A.S., ed. - **Readings in Human-Computer Interaction: A Multidisciplinary Approach.** California, Morgn Kaufmann, 1987.

[RM86] RUMELHART, D.E.; McCLELLAND, J.L.; The PDP Research Group - **Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.** 5.printing. MIT Press, 1986. 547p.

[RM86a] RUMELHART, D.E.; McCLELLAND, J.L.; The PDP Research Group - **Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models.** 5.printing. MIT Press, 1986. 611p.

[Ros58] ROSENBLATT, F. - The Perceptron, a Probabilistic Model for Information Storage and Organization in the Brain. **Psycological Review., 62:559ff,** 1958.

[RZ86] RUMELHART, D.E. & ZIPSER, D. - Feature Discovery by Competitive Learning. In: RUMELHART, D.E. & McCLELLAND, J.L. ed. - **Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.** 5.printing. MIT Press, 1986. p.151-193.

[SR87] SEJNOWSKI, T.J. & ROSENBERG, C.R. - Parallel netwoks that learn to pronounce English text. **Complex Syst., 1:145-168,** 1987.

[SS90] SEJNOWSKI, T.J. & STANTON, P.K. - Covariance Storage in the Hippocampus. In: ZORNETZER, S.F.; DAVIS, J.L.; LAU, C., eds - **An Introduction to Neural and Eletronic Networks.** Academic Press, Inc., 1990. p.365-377.

[Ste92] STEPANKOVA, O. - An Introduction to Qualitative Reasoning. **Lectures Notes in Artificial Intelligence., 617:404-439,** 1992.

[Ster90] STERNBERG, R.J. - The computational metaphor. In: _____ - **Metaphors of mind: Conceptions of the nature of intelligence,** Cambridge University Press, 1990. p.113-161.

[Ten90] TENORIO, M.F.M. - Topology synthesis networks: self organization of structure and weight adjustment as a learning paradigm. **Parallel Computing., 14:363-380,** 1990.

[Tur92] TURBAN, E. - Neural Computing and AI. In: _____ - **Expert Systems and Applied Artificial Intelligence.** Macmillan Publishing Company, 1992. p.621-663.

[Vem88] VEMURI, V. - Artificial Neural Networks: An Introduction. In: VEMURI, V., ed. - **Artificial Neural Networks: Theorical Concepts.,** Computer Society Press of IEEE, 1988. p.1-12.

[Wer74] WERBOS, P.J. - **Beyond regression: New tools for prediction and analysis in the behavioral sciences.** Cambridge, MA, 1974. [Ph.D. Thesis - Harvard University]

[WF88] WALTZ, D. & FELDMAN, J.A. - Connectionist Models and Their Implication In: WALTZ, D. & FELDMAN, J.A., ed. - **Connectionist Models and Their Implications: Readings From Cognitive Science,** Ablex Pulishing Corporation, 1988. p.1-11.

[WH60] WIDROW, B. & HOFF, M.E. - Adaptive Switching Circuits. **WESCON Convention Record 4.,** 1960.

[Win92] WINSTON, P.H. - Learning by Training Neural Nets. In: _____ - **Artificial Intelligence**, Third Edition, Addison Wesley, 1992, p.443- 469.

APÊNDICE I

Definições Matemáticas

AI.1 - DERIVADA DIRECIONAL.

Seja f uma função de duas variáveis e $\vec{\alpha} = (\cos\theta, \sin\theta) = \cos\theta\,\vec{i} + \sin\theta\,\vec{j}$ uma direção qualquer no plano O_{xy}

Chama-se derivada de f na direção $\vec{\alpha}$, no ponto $\vec{P}$ = *(x,y)*, ao seguinte limite, quando existir, indicado pelos símbolos $\partial f/\partial\vec{\alpha}$, $\nabla_{\vec{\alpha}} f$ ou $D_{\vec{\alpha}} f$ [Avi87].

$$\frac{\partial f}{\partial \vec{\alpha}}(\vec{P}) = (\nabla_{\vec{\alpha}} f)(\vec{P}) = \lim_{r\to 0} \frac{f(\vec{P} + r\vec{\alpha}) - f(\vec{P})}{r} \qquad \text{(A)}$$

que também pode ser escrito como

$$\frac{\partial f}{\partial \vec{\alpha}}(x,y) = \lim_{r\to 0} \frac{f(x + r\cos\theta, y + r\sin\theta) - f(x,y)}{r} \qquad \text{(B)}$$

Podemos então entender a derivada direcional de f numa dada direção como sendo o limite do quociente $(\Delta f/\Delta s)$ da variação Δf pela distância Δs percorrida na direção dada $\vec{\alpha}$, quando Δs tende a zero [Kap82].

Geometricamente podemos entender a derivada direcional de uma função *f(x,y)* numa direção $\vec{\alpha} = (\cos\theta, \sin\theta)$ e num ponto $P_0(x_0, y_0)$ como o declive da curva de interseção da superfície $z = f(x,y)$ com um plano que contenha P_0 e que seja paralelo ao eixo O_z e à direção $\vec{\alpha}$.

Sendo Ψ o ângulo dessa interseção com o plano horizontal O_{xy}, então

$$D_{\vec{\alpha}}(\vec{P}_0) = \tan\Psi \qquad \text{(C)}$$

Quando a direção escolhida $\vec{\alpha}$ for paralela ao eixo x ou ao eixo y teremos então as derivadas parciais respectivamente $\partial\alpha/\partial x$ e $\partial f/\partial y$

AI.2 - GRADIENTE.

Seja f uma função de duas variáveis, $f = f(x,y)$ definida num certo domínio do espaço. Se suas derivadas primeiras existirem neste domínio, então definimos o vetor gradiente de f, indicado pelos símbolos grad f e ∇f, como sendo:

$$\nabla f = \text{grad}\, f = \frac{\partial f}{\partial x}\bar{i} + \frac{\partial f}{\partial y}\bar{j} \qquad \text{(D)}$$

O vetor gradiente também pode ser definido como o vetor que satisfaz a seguinte condição:

$$D_{\bar{\alpha}} f = \|\nabla f\| \cos\phi \qquad \text{(E)}$$

onde ϕ é o ângulo entre os vetores ∇f e $\bar{\alpha}$.

Podemos ver de (E) que a derivada direcional de f atinge o seu valor máximo, supondo-se $\nabla f \neq 0$, quando $\phi = 0$ $(\cos 0 = 1)$, isto é, quando a direção $\bar{\alpha}$ coincide com a direção do gradiente de f. Isto significa que o gradiente aponta para a direção em que f cresce mais rapidamente.

Observamos também que a derivada direcional é zero quando a direção $\bar{\alpha}$ é perpendicular à direção do gradiente $(\cos \pi/2)$ [Avi87].

Embora tenhamos definido f como uma função de duas variáveis (para facilitar a interpretação geométrica) os conceitos de derivada direcional e gradiente podem ser generalizados para funções com várias variáveis.

AI.3 - REGRA DA CADEIA.

Suponha que y seja uma função de várias variáveis intermediárias x_i e que cada x_i seja uma função de uma variável t e que ainda as funções consideradas seja definidas em domínios apropriados e possuam derivadas primeiras contínuas.

Então podemos obter a derivada de y em relação a t, dy/dt, utilizando-se a regra da cadeia expressa por:

$$\frac{dy}{dx} = \sum_i \frac{\partial y}{\partial x_i}\frac{dx_i}{dt}$$
$$= \frac{\partial y}{\partial x_1}\frac{dx_1}{dt} + \frac{\partial y}{\partial x_2}\frac{dx_2}{dt} + \ldots + \frac{\partial y}{\partial x_n}\frac{dx_n}{dt}$$
$$= \frac{dx_1}{dt}\frac{\partial y}{\partial x_1} + \frac{dx_2}{dt}\frac{\partial y}{\partial x_2} + \ldots + \frac{dx_n}{dt}\frac{\partial y}{\partial x_n}$$
$$= \sum_i \frac{dx_i}{dt}\frac{\partial y}{\partial x_i} \qquad \text{(F)}$$

AI.4 - MÉTODO DO GRADIENTE DESCENDENTE.

O método do gradiente descendente ("steepest descent") , é um dos mais antigos e conhecidos métodos para minimizar uma função de várias variáveis.

Como todo algoritmo dito descendente, o método do gradiente descendente segue uma estrutura onde:

- Começa-se com um ponto inicial do domínio da função.
- Determina-se de acordo com uma regra fixada uma direção de movimento.
- Move-se naquela direção até se encontrar um mínimo (relativo) da função.
- Esse mínimo torna-se o novo ponto inicial.
- Repete-se o processo.

O processo de determinação do ponto mínimo em uma dada direção é chamado de busca linear. Em geral esse mínimo para funções não lineares não pode ser determinado analiticamente, sendo encontrado por uma busca inteligente ao longo da direção.

A busca de um mínimo em um problema com várias dimensões reduz-se então a sucessivos problemas de se encontrar mínimos ao longo de uma reta.

Seja f, a função a minimizar, uma função tendo suas derivadas primeiras contínuas no espaço E_n e seja $\vec{g}(\vec{x})$ um vetor coluna n dimensional tal que

$$\vec{g}(\vec{x}) = \nabla f(\vec{x})^T \qquad \text{(G)}$$

O método do gradiente descendente é definido por um algoritmo iterativo

$$\bar{x}_{K+1} = \bar{x}_K - \alpha_K \bar{g}_K \qquad \text{(H)}$$

onde α_K é um escalar não negativo que minimiza $f(\bar{x}_K - \alpha \bar{g}_K)$, isto é, minimiza a função f na reta que passa por x_K e que tenha a direção de $-\bar{g}_K$.

Nesse ponto mínimo teremos $\bar{x}_{K+1}$.